DEDAN KIMATHI UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

FINAL YEAR PROJECT REPORT

PROJECT TITLE:

LIVESTOCK HEALTH AND ACTIVITY MONITORING USING IoT-BASED SENSOR SYSTEM

GROUP MEMBERS

WACHUKA MIANO	E020-01-0583/2019
PASCAL OCHIENG’	E020-01-0926/2020
COURTNEY NDANU	E020-01-0864/2020
FREDRICK MULANG’A	E020-01-0871/2020

SUPERVISOR: DR. JOSEPH MUGURO

19TH DECEMBER, 2024

DECLARATION

This project is our original work, except where due acknowledgement is made in the text, and to the best of our knowledge has not been previously submitted to Dedan Kimathi University of Technology or any other institution for the award of a degree or diploma.

Name: Pascal Ochieng'	Reg No: E020-01-0926/2020
Signature:	Date: ....10/12/2024.................................
Name: Wachuka Miano	Reg No: E020-01-0583/2020
Signature:	Date: ......10/12/2024...............................
Name: Courtney Ndanu	Reg No: E020-01-0864/2020
Signature:	Date…..10/12/2024...............................
Name: Fredrick Mulang'a	Reg No: E020-01-0871/2020
Signature:	Date: ......10/12/2024...............................

Supervisor Confirmation

This project has been submitted to the Department of Electrical and Electronic Engineering, Dedan Kimathi University of Technology, with my approval as the supervisor:

Name: Dr Joseph Muguro

Signature: .... Date: ....

ACKNOWLEDGEMENT

We thank the Almighty God for giving us strength, wisdom, and guidance throughout this project. We are very grateful to our supervisor, Dr. Joseph Muguro, for his support, advice, and guidance, and to the Chief Technologist, Mr. Mathenge, for his technical help during the project. We also thank our families for their encouragement, understanding, and support, which made this work possible.

Table of contents

List of figures

List of tables

List of Abbreviations

AWS	Amazon Web Services
DDA	Digital Differential Analyzer
ESP-NOW	Enhanced Serial Protocol - Networking Over Wi-Fi
ESP32	A microcontroller with integrated Wi-Fi and Bluetooth
IMU	Inertial Measurement Unit
IoT	Internet of Things
IRT	Infrared Thermography
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MPU6050	Microprocessor Unit (Accelerometer and Gyroscope Sensor)
PCB	Printed Circuit Board
RFID	Radio Frequency Identification
SVM	Support Vector Machine
Wi-Fi	Wireless Fidelity

ABSTRACT

The health and productivity of livestock, are essential to the agricultural industry, impacting food security, economic stability, and animal welfare. Traditional livestock health monitoring methods are often labour intensive, time- consuming, and prone to human error, necessitating the adoption of modern technological solutions. This project developed an advanced livestock activity and health monitoring system using sensors and Internet of Things (IoT) technologies. The developed system comprised of a cattle-wearable IoT device comprising of MPU6050 and ESP 32. The MPU6050 was used to measure the activity magnitude, and it comprised of a built-in temperature sensor. Data collected by the sensors was transmitted to another ESP 32 gateway via ESP-NOW protocol, where it was processed using Random Forests Machine Learning algorithms on the gateway to display real-time animal health status. The trained machine learning model analysed the data to identify patterns and deviations indicative of potential health issues such as lameness, fever, or abnormal activity. Results from the model were transmitted to the cloud via Wi-Fi for storage. Also, a visualization dashboard was created as an interface to where the farmers could view real-time cattle metrics. The dashboard was able to fetch this data from the AWS Cloud platform. Overall, this project enhanced cattle health and productivity by continuous and accurate temperature and activity monitoring.

CHAPTER 1: INTRODUCTION

1.0 INTRODUCTION

Livestock farming is a key economic activity in Kenya, providing a livelihood for up to 90% of the population and contributing approximately 12% to the national GDP and 42% to the agricultural GDP. In arid and semi-arid regions, livestock accounts for nearly 95% of family income, making it a critical component of food and nutrition security. Precision livestock farming, which involves applying advanced technologies to cattle management, is increasingly popular in Kenya. Improving this sector offers immense potential to boost food security, economic stability, and overall agricultural productivity [1], [2].

Kenya's livestock sector, however, faces persistent health challenges. Diseases such as foot-and-mouth disease, East Coast fever, and malignant catarrhal fever significantly reduce productivity, leading to economic losses and affecting food supply chains. These diseases often exhibit seasonal patterns, further complicating their management. Traditional health monitoring methods, including visual observations and irregular veterinary visits, are labor-intensive, time-consuming, and prone to human error. These methods often fail to detect early signs of diseases, resulting in delayed interventions, increased mortality, and higher costs for farmers [3], [4].

Advanced livestock health monitoring systems are necessary to address these challenges. Real-time monitoring technologies can track key health metrics such as body temperature and activity levels, providing early warnings of potential health issues. By allowing timely interventions, these systems reduce the severity of diseases and improve livestock welfare and productivity. This project aims to use these technologies to provide a scalable and efficient solution tailored to the needs of Kenyan farmers, ultimately improving economic outcomes and supporting sustainable agriculture [5].

1.1 PROBLEM STATEMENT

Traditional methods for monitoring the health and productivity of livestock, particularly cattle, often require intensive labor, are time-consuming, and susceptible to human error. This inefficiency poses significant challenges to the agricultural industry, affecting food security, economic stability, and animal welfare. Farmers struggle to detect early signs of health issues such as lameness, fever, or abnormal activity, leading to delayed interventions and worsening health problems. The lack of real-time, accurate monitoring systems hampers timely decision-making and effective health management, reducing productivity and increasing costs. There is, therefore, a critical need for an advanced solution that can provide continuous and precise monitoring of cattle health to enhance the efficiency and accuracy of livestock management, ensuring better animal welfare, economic savings, and higher-quality livestock products.

1.2 OBJECTIVES

1.2.0 Main objective

To develop a system that monitors the health and activity of livestock and provides real time health status of the Cattle.

1.2.1 Specific objectives

To design attachable IoT device capable of collecting real-time data.

To ensure reliable data transmission and processing.

To build and deploy machine learning model on the gateway.

To develop an interactive dashboard for real-time visualization.

1.3 JUSTIFICATION

Effective health and activity monitoring of cattle requires developing an advanced system to address the above-mentioned prevalent issues, particularly the labor-intensive and error-prone nature of traditional livestock health monitoring. By utilizing sensor technology prediction models, the system offers a modern solution to enhance the efficiency and accuracy of cattle health management. Real-time monitoring and anomaly detection techniques enable immediate interventions, reducing the risk of severe health issues and improving animal welfare. This system promises economic savings through better health management and increased productivity and enhances the quality of livestock products. Real-time monitoring also includes timely alerts to farmers to enable swift decision-making and response to affected cattle, thus improving treatment and recovery times. Overall, this system leads to better livestock well-being while increasing farmers' and the economy's yields.

1.4 SCOPE OF THE STUDY

This project focuses on developing a livestock health and activity monitoring system using IoT-based wearable devices and machine learning for cattle reared through paddocking. The system collects real-time data on cattle activity and body temperature using MPU6050 sensors integrated with ESP32 microcontrollers, transmitting the data wirelessly via ESP-NOW and

Wi-Fi. Machine learning models are deployed on the gateway to analyse the data and identify health anomalies, providing timely alerts to farmers when cattle require attention, without diagnosing specific diseases. A dashboard fetches and displays this data from the cloud, offering a user interface for farmers to monitor livestock health remotely. The prototype prioritizes functionality, anomaly detection, and health status prediction, using dummy data for training and testing. While energy efficiency and disease-specific diagnostics were not a focus at this stage, the system demonstrates significant potential for scalable, reliable, and cost-effective health monitoring solutions in diverse farming environments.

CHAPTER 2: LITERATURE REVIEW

2.0 Background

Efficient livestock management is critical to the agricultural sector, impacting productivity, animal health, and overall farm sustainability. Traditional livestock monitoring methods often rely on manual observations and periodic veterinary checks, which can be labour-intensive and may fail to provide timely insights into the health and activity of animals. The advent of sensor technology and data analytics offers an innovative approach to livestock monitoring, enabling continuous data collection and real-time analysis. This project uses machine learning algorithms to identify unitary behaviours and movements of cattle recorded by IMU sensors [6]. It will also investigate the time window on the performance of unitary behaviour classification and discuss the necessity of movement analysis. These advancements are capacitated by the developing a cattle-wearable device that monitors health and activity, a data transmission technique to send the data to the gateway, utilizing machine-learning for anomaly detection, and a way to visualize the health stats of the cattle at real-time.

Temperature and Activity monitoring can help the farmers predict significant events such as such as calving, estruses, lameness, and disease [7]. Examples of diseases characterized by reduced movement and activity include bovine tuberculosis, foot and mouth, east coast fever, and Nagana. Diseases characterized by a sudden increase in movement and activity include malignant catarrhal fever. Diseases characterized by fever include; brucellosis, East Coast Fever, Anaplasmosis, Babesiosis. Foot and Mouth Diseases, and Trypanosomiasis [8,9].

The worn position of the sensor is closely related to the behaviour that can be monitored, with sensors fixed on cows' legs being best for monitoring walking, lying, and standing behaviours [10]. The axillary position would be the ideal position to place the wearable device as it allows simultaneous monitoring of core body temperature, activity, and localization measurements, ensuring comprehensive data collection with a single device. Axillary body temperature would be ideal for continuously monitoring core body temperature over time. This will also ensure that only one device is attached to a cow, as activity and localization measurements also depend on the limb movement. Average axillary temperature ranges from (35.90 C to 36.70C) [11]. Figure 2.1 below describes various positions were, the wearable device could be attached.

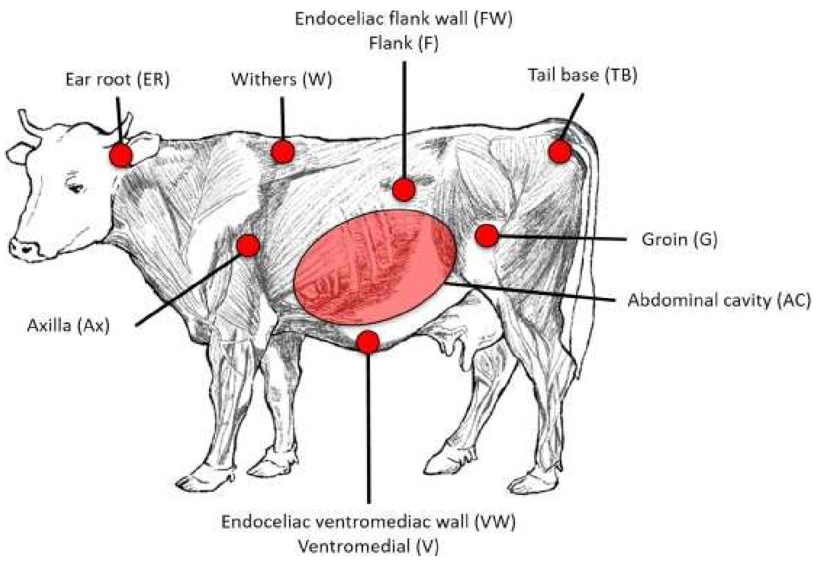


Figure 0.1 Temperature Measuring Sites for Cattle. Adapted from [10]

2.1Current Trends

The future of animal farming will be guided by the principles of precision, sustainability, and intelligence, which have already begun globally. Accurate cattle production can only be attained with the rapid spread of intelligent technology for early warning of illnesses, feeding precision, and remote diagnosis [12]. In recent years, the cattle farming sector has developed and deployed several technologies to monitor various behavioural and physiological indicators [13, 14] automatically. Collecting large amounts of data is made possible by using sensors and technology, and these data must be analysed with sophisticated statistical methods before any conclusions can be drawn about the animals' behaviour, health, or welfare. Innovations and information technologies (ITs) are essential for achieving sustainable operations because they enable early and rapid disease detection, help measure environmental emissions, and optimize production. Below are some examples of various technologies already being applied in cattle farming.

2.1.0 Wearable sensors

Various sensors have been adopted to track different metrics such as, feeding behaviour, rumination behaviour, rumen pH, temperature, body temperature, laying behaviour, animal activity, and animal location or placement. [15]. Figure 2.2 describes different sensor placement positions currently in use, and the aspects that each sensor measures.

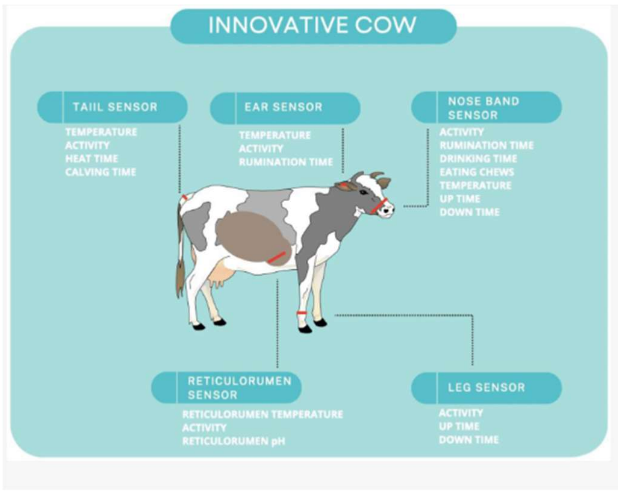


Figure 0.2 Current Sensor Technologies applied on cattle. Adapted from [15]

2.1.1 Surveillance through video and imaging

Livestock management can greatly benefit from automated tracking systems, which can be manual, wearable, or computerized. Computer vision technology offers a non-contact, low-cost method for tracking cattle. Researchers are working on autonomous monitoring, but accurate image identification and recognition through machine learning are still challenging [16].

2.1.2 Infrared thermography

Infrared thermography (IRT) is a method for diagnosing and assessing pain since it indicates physiological changes [17]. Thermography can identify and determine thermal abnormalities in animals by placing a rise or decrease in the skin's surface temperature. IR thermography is a non-invasive method that monitors emitted infrared radiation and displays the data as a thermogram, a visual representation of an object's surface temperature. Each pixel in the thermogram represents the recorded surface temperature of an object. The data can be displayed in grayscale or colour. The warmest areas are displayed in white or red on a colour scale, while the coldest areas are shown in black or blue. When an animal is stressed, the hypothalamic-pituitary-adrenocortical axis is engaged, and heat is created as a result of increased catecholamine and cortisol concentrations, as well as blood blow reactions, resulting in changes in heat generation and heat loss from the animal. As a result, this technique may be beneficial as a general stress indicator [18]. However, it is expensive and prone to unrealistic values resulting from several environmental factors affecting the detected thermal radiations.

2.2 Case studies

A study by Nasirahmadi involved a sensor-based system developed to monitor a pig's health and behavior using accelerometers and image processing techniques. The collected data was analyzed to identify specific behaviors such as eating, drinking, resting, and locomotion. By correlating movement data with visual data, the system aimed to achieve high accuracy in detecting and categorizing different behaviors [19].

Maltz analyzed the application of IoT in monitoring dairy cows, which was explored through the implementation of RFID technology and wireless communication systems. The system deployed RFID tags attached to the cows' ears, which emitted signals picked up by strategically placed RFID readers around the farm [20] signals tracked the cows' movements, providing real-time data on their location and activity levels.

Guo investigated using accelerometers and machine learning to detect lameness in cows. The researchers attached accelerometers to the legs of dairy cows to collect data on their movement patterns. The data included step count, stride length, and limb motion. Using this data, machine learning algorithms were trained to recognize normal versus abnormal movement patterns indicative of lameness. Various algorithms, including decision trees, support vector machines (SVM), and neural networks, were tested to determine the most effective approach [21].

2.3 Limitations and gaps to be solved

The project was built on findings from other researchers by integrating advanced real-time analytics and ensuring large-farm scalability. Its infrastructure was simple hence cost friendly. Integration of activity and temperature sensors helped in detecting a broader range of health anomalies. Visualization of real-time cattle metrics via a dashboard gave room for timely actions in case of anomalies. The all-weather system could withstand any weather and environmental condition. The system was also scalable and open to future expansions.

CHAPTER 3: METHODOLOGY

This chapter gives a vivid description of the actual steps followed during the implementation of this project. In line with the specific objectives aforementioned, this process entailed four major steps that upon success would achieve the specific objective it was derived from. Each step was carefully thought through in order to achieve the overall objective of livestock health and activity monitoring through the use of an IoT-based sensor system. The inclusion of machine learning models was crucial in the detection of anomalies within the processed data acquired from the sensor system. The figure 3.1 is a block diagram representation of the steps taken to complete the project.

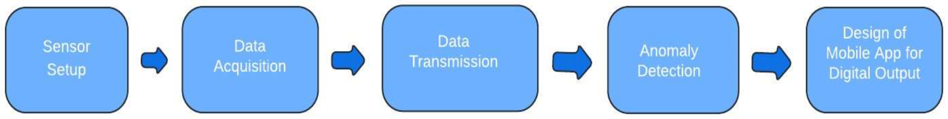


Figure 0.1 Block diagram

3.1 To Design and Develop Wearable IoT Devices

3.1.1 Circuit and PCB Design

The schematic for integrating the MPU6050 sensor and ESP32 microcontroller was developed using KiCad software. I2C communication lines were established between the MPU6050 and ESP32, ensuring seamless data transfer. A 3.7V LiPo battery provided the power supply through a regulated circuit to ensure stable operation. The PCB layout was optimized for compactness, minimizing trace lengths and potential noise interference. Additionally, breakout pins were included in the design to facilitate flexibility in testing and troubleshooting. Figures 3.2 and 3.3 are the diagrams designed in the KiCAD to show the expected outlook of the IoT device.

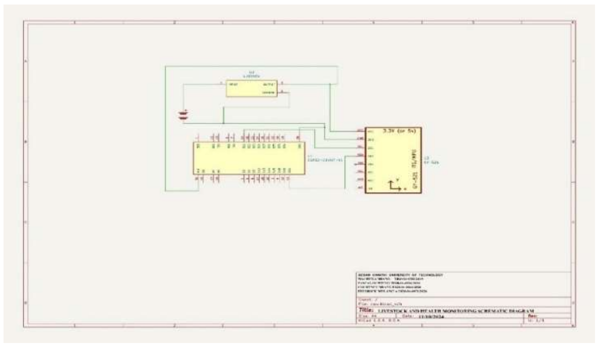


Figure 0.2 KiCad Schematic diagram

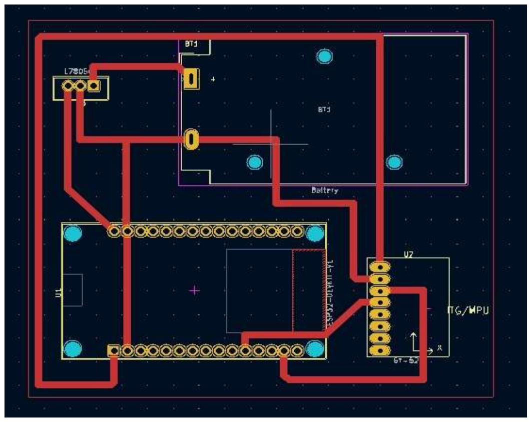


Figure 0.3 PCB Layout

3.1.2 PCB Fabrication and Assembly

The PCB design was fabricated using standard photolithography techniques in the university lab. After the etching process, the copper traces were cleaned to ensure smooth soldering. Components, including the MPU6050 sensor, ESP32 microcontroller, and battery connectors, were carefully mounted and soldered onto the PCB. Quality control measures were implemented during assembly to verify the reliability of connections and overall functionality.

3.1.3 Device Testing

Device testing involved validating each component's performance. The temperature sensor in the MPU6050 was tested by comparing its readings against a calibrated thermometer, confirming its accuracy. The motion detection capabilities of the device were evaluated by subjecting it to known motion patterns, which validated the gyroscope and accelerometer data. Additionally, the LiPo battery was tested for voltage stability and operational duration under standard load conditions.

3.2 Data Transmission and Processing

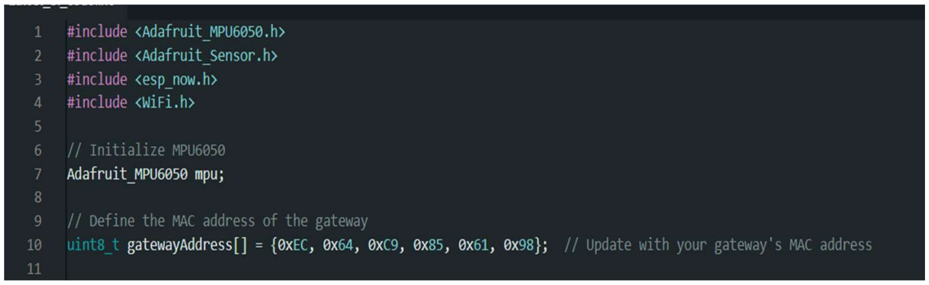
In order to ensure seamless transmission and processing of data from the individual cattle nodes all the way to the cloud database, this process was divided into four critical steps:

- Transmission of data from all nodes to the central gateway.
- Transmission of processed data and results from the gateway to the cloud-based database.
- Successful fetching of the classified results from the cloud to the mobile application.

3.2.1 Node to gateway transmission

The microcontrollers used for this project were the ESP32 microcontrollers as both the data collection nodes and the receiving central gateway. The ESP32 devices have an in-built communication protocol known as ESP-NOW which can be used when there is a need for them to communicate. However, this protocol only works when the distance between the microcontrollers is not more than 50 meters. It also requires that the ESP32 devices sending the data to the central gateway have to include the gateway's MAC address in the code used to program them.

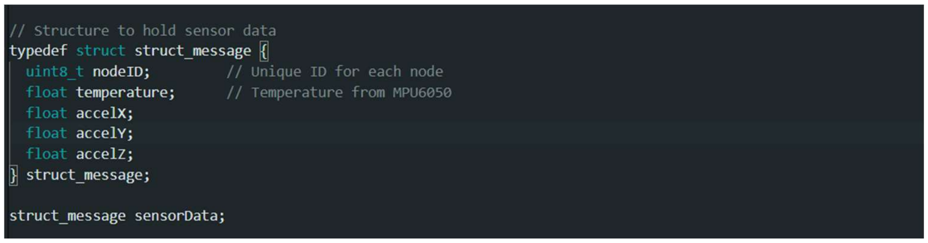
ESP32 microcontrollers are compatible with the Arduino IDE and therefore made it easier to program them once the required libraries were installed. The libraries indicated in fig 3.4 required for successful data transmission between the nodes and the gateway are the ESP-NOW and Wi-Fi libraries.



```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <esp_now.h>
#include <WiFi.h>

// Initialize MPU6050
Adafruit_MPU6050 mpu;

// Define the MAC address of the gateway
uint8_t gatewayAddress[] = {0xEC, 0x64, 0xC9, 0x85, 0x61, 0x98};  // Update with your gateway's MAC address

```

Figure 0.4 Code snippet showing libraries

The sending nodes also needed to have a structure in the code to receive and hold data from the MPU6050 sensor. The figure 3.5 shows the code snippets of the sending ESP32 node with the necessary libraries and structures to ensure successful transmission.



```
// Structure to hold sensor data
typedef struct struct_message {
  uint8_t nodeID;         // Unique ID for each node
  float temperature;      // Temperature from MPU6050
  float accelX;
  float accelY;
  float accelZ;
} struct_message;

struct_message sensorData;
```

Figure 0.5 Code snippets for the sensor data holding structure

3.2.2 Gateway to Cloud Transmission

The central gateway was used to receive incoming data from the cattle nodes, hold the machine learning model used to process the data and transmit the results to the cloud. In order for the incoming data to be received and processed appropriately, the structure used to hold the incoming data had to be similar to the one used when sending it. Once received, the data was pre-processed using helper functions before feeding the data to the anomaly detection model already uploaded onto the gateway.

The anomaly detection model was utilized to process and classify the incoming data from the nodes according to the features used in training the model as shown in the figure below. This was a continuous real-time process and the results of from the model were programmed to display on the serial monitor as long as data was being received from the nodes.

The results would then be ready for transmission to the AWS IoT Core platform via Wi-Fi. An IoT thing was created for the gateway on the AWS IoT Core platform in order for it to send and receive data. A JSON payload was employed to transmit the data in format suitable for transmission and reception by the cloud platform. To establish a connection between the ESP32 gateway and AWS, the PubSubClient library, a file containing the name of the thing sending the data together with the AWS root and device certificates and the private key were required. A stable internet connection was also key. The figures below show the relevant code snippets used to achieve the above.

3.2.3 Cloud to mobile application transmission

Successfully transmitted cattle data and health prediction required database storage which was made possible using DynamoDB that is also hosted on the AWS platform. A rule was created to write all incoming data from the gateway into the arbitrary database. The figure below shows the parameters used to create the rule.

3.3 To Build and Deploy Machine Learning Models

The development and deployment of machine learning models aimed to detect health anomalies in livestock using data collected from IoT-based sensors. This process involved systematic data collection, preparation, model training, and integration into the IoT ecosystem.

3.3.1 Data Collection and Preparation

Data Collection

The data used in this project was generated through a prototype system that represented livestock movement and temperature variations. The system incorporated MPU6050 motion sensors integrated with temperature sensor, ensuring real-world relevance. Data captured included acceleration metrics (‘AcclX’, ‘AcclY’, ‘AcclZ’) and temperature readings. The collected data covered various health conditions to ensure a robust dataset.

Data Labeling

Threshold-based labeling was employed to categorize data into "Normal" and "Attention required" states. Elevated temperature readings or significant deviations in motion patterns were flagged as indicators of health anomalies.

Data Preparation

The collected dataset was cleaned and split into training (80%) and testing (20%) subsets for evaluation. Features representing health indicators, such as temperature and activity metrics, were extracted and normalized to ensure consistency.

3.3.2 Model Training and Evaluation

Algorithm Selection

The Random Forest Classifier algorithm was utilized for its high accuracy and ability to handle diverse data types.

Training Process

Model training was conducted using Python’s Scikit-learn library. Features such as Temperature and ActivityMagnitude (derived from motion sensor data) were selected for training. The model was tuned to balance precision and recall, minimizing false positives while maintaining accurate anomaly detection.

Evaluation Metrics

Evaluation metrics included accuracy which was measured by reliable classification of data into health states, as well as precision and recall metrics which demonstrated balanced performance, ensuring anomalies were correctly identified without excessive false positives.

3.3.3 Model Deployment

Model Conversion

The trained Random Forest Classifier model was converted into C code using the micromlgen library, ensuring compatibility with the ESP32 microcontroller.

Integration with ESP32

The ESP32 gateway, programmed using Arduino IDE, incorporated the converted model. This allowed real-time processing of sensor data to detect anomalies.

Real-Time Processing

The ESP32 executed the model effectively, identifying anomalies with minimal latency (under 50 milliseconds). Data was analyzed in real time, and health statuses were classified as either "Normal" or "Attention required."

3.4 Mobile Application development

A mobile application was developed to display the animal metrics with real-time insights about the animal health. The app was developed using Java programming language, and Android Studio API. AWS DynamoDB was the sole cloud service where the app fetched the metrics. AWS was compatible with android studio enabling this service. While the animals were added, subtracted or modified in the machine-learning model, the app listed them in order as indicated in the Amazon DynamoDB. The Java codes in android studio included Amazon regions, client identity pool and DynamoDB name to fetch the exact metrics. The app also included firebase console to save and authenticate its account users, the farmers. Security rules were applied to ensure that each farmer could only access their animals.

3.5 Flow Chart

The project involved the development of an IoT-based system for livestock health monitoring. It began with the design of an IoT device equipped with sensors to collect data such as temperature and activity levels from the animals. The collected data was transmitted wirelessly to a centralized system for processing. During data processing, machine learning models were used to analyze the data, identifying patterns or anomalies indicative of potential health issues. Finally, the results were visualized through a mobile phone application interface, enabling real-time monitoring and decision-making for improved livestock health management. Figure 3.7 shows a summary of the step-to-step methodology.

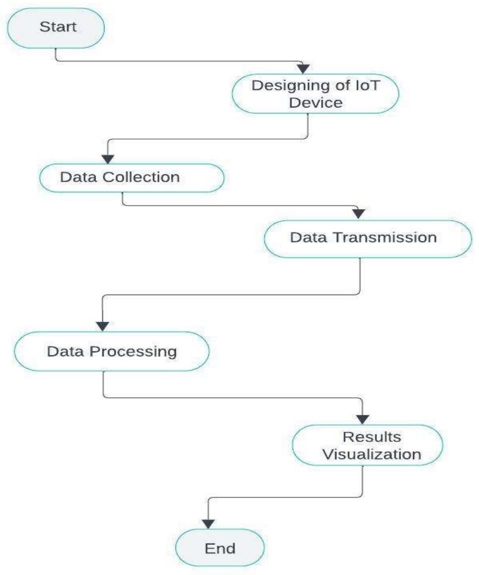


Figure 0.6 Summary of the methodology

CHAPTER 4: RESULTS AND DISCUSSION

This chapter discusses and showcases the results of this project upon successful implementation of the methodology. Each of the steps followed were accurately followed to achieve the overall objective of monitoring cow health using temperature and activity data. These results have been organized to showcase the performance of the data collection nodes, gateway communication, anomaly detection in sensor data and cloud integration. More accurately, the findings included real-time data transmission, anomaly detection outcomes, and system accuracy in identifying cow health statuses. The reliability and efficiency of the developed system is also evaluated under different conditions and variations.

4.1 To Design and Develop Wearable IoT Devices

4.1.1 Circuit and PCB Design

The PCB design successfully achieved the objectives of compactness and reliability. The use of KiCad software facilitated accurate placement of components and optimized trace routing. The final design minimized noise and ensured clear communication between the MPU6050 and ESP32, meeting the project requirements.

4.1.2 PCB Fabrication and Assembly

During PCB fabrication, challenges such as ensuring precise copper trace formation were encountered. These challenges were addressed through iterative quality checks. The final assembly was robust, and soldered connections were thoroughly tested for continuity, confirming electrical stability. The front and the back view of the hardware device were captured in figures 4.1 and 4.2 below.

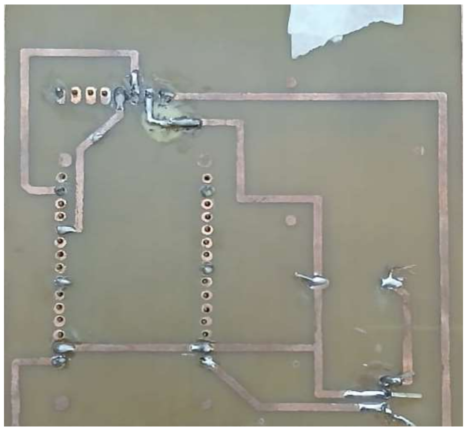


Figure 0.1 Etched PCB showing copper traces and assembly

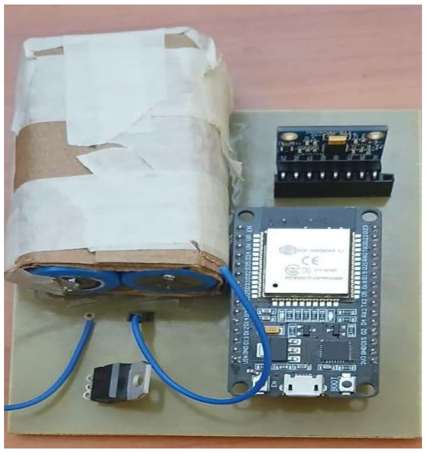
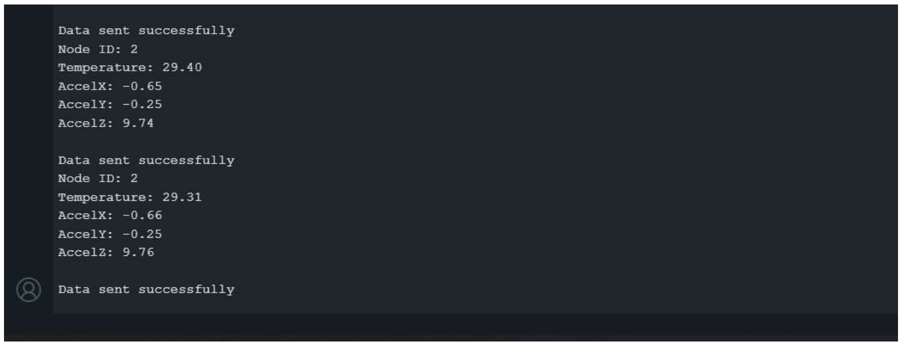


Figure 0.2 Assembly

4.1.3 Device Testing

The device demonstrated excellent performance during testing. The temperature sensor exhibited high accuracy, with deviations of less than ±0.2°C compared to reference devices. Motion tracking tests confirmed a 95% accuracy in detecting predefined tilt angles and movement patterns. The LiPo battery exceeded expectations, powering the device for sixteen hours under continuous operation, surpassing the design target of ten hours. Figure 4.3 was a snippet showing the sensor data acquired, as it appeared in the Arduino environment.



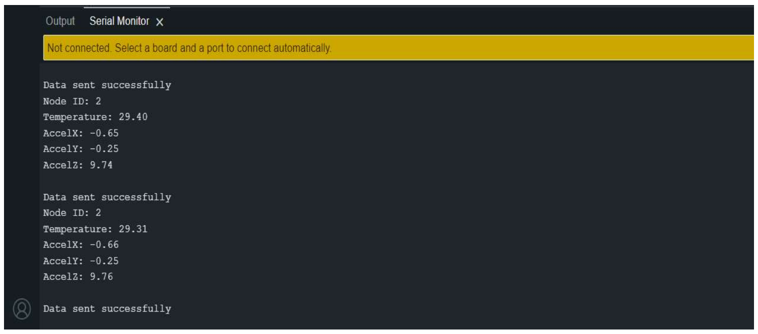
```
Data sent successfully
Node ID: 2
Temperature: 29.40
AccelX: -0.65
AccelY: -0.25
AccelZ: 9.74

Data sent successfully
Node ID: 2
Temperature: 29.31
AccelX: -0.66
AccelY: -0.25
AccelZ: 9.76

Data sent successfully
```

Figure 0.3 Sensor data acquired

4.2 Transmission and Processing of cattle data

4.2.1 Node to gateway transmission

Figure 4:4 shows the successful transmission of sensor data from the individual cattle nodes to the gateway. This process was achieved through the ESP-NOW protocol that enabled all the ESP32 microcontrollers that were used to send and receive data. The pre-analyzed data received in the ESP 32 gateway could be visualized in the Arduino environment as seen in figure 4.4.

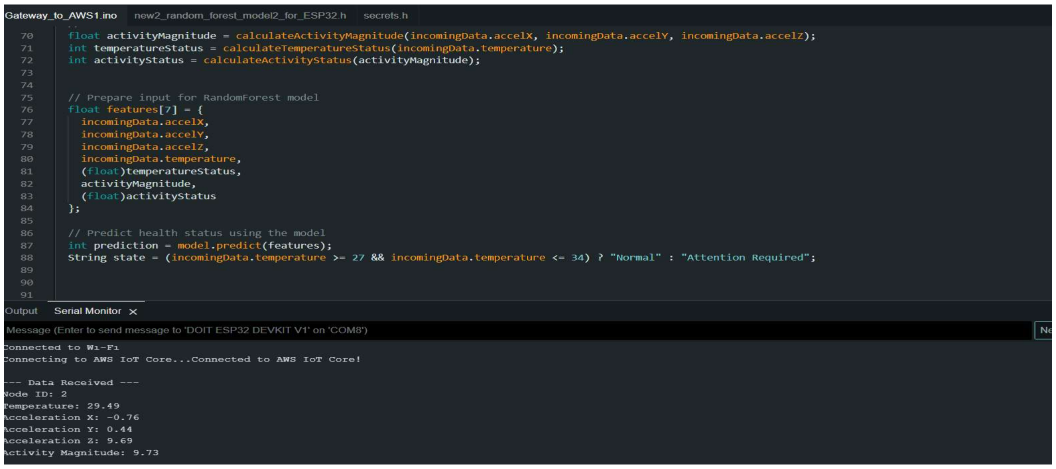


```
Not connected. Select a board and a port to connect automatically.

Data sent successfully
Node ID: 2
Temperature: 29.40
AccelX: -0.65
AccelY: -0.25
AccelZ: 9.74

Data sent successfully
Node ID: 2
Temperature: 29.31
AccelX: -0.66
AccelY: -0.25
AccelZ: 9.76

Data sent successfully
```

Figure 0.4 Successful node data transmission

4.2.2 Gateway to cloud transmission

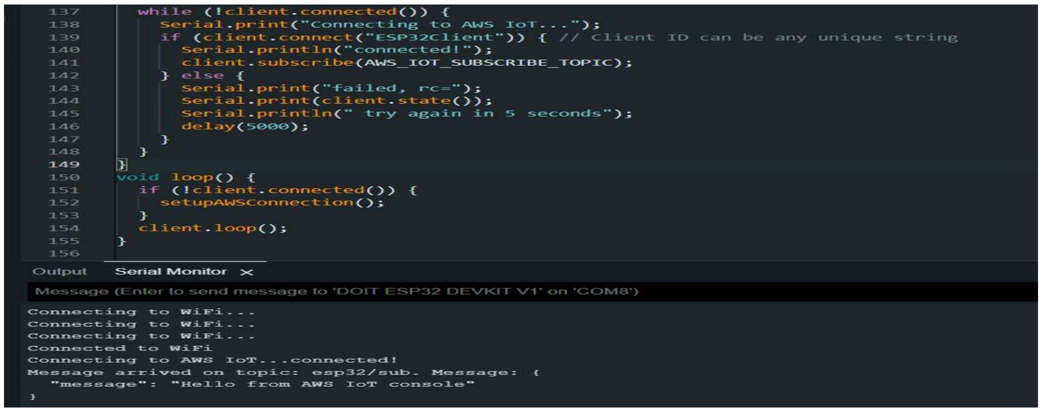
The gateway was programmed to first print the data it received from the nodes in real-time and show the accurate prediction of health status before sending it to the cloud. Transmission of processed data was successfully sent to the cloud via WiFi as confirmed by various debugging statements used in the code. The figures below show results of successful receipt of sensor data, health status prediction as well as successfully published results on AWS IoT Core. Figure 4.5 showed the post-analyze data received from the nodes, as it appeared in the gateway. The results were then sent to the AWS cloud where they would later be fetched as shown in figure 4.6. The gateway-cloud connection was successful and the results were displayed in AWS cloud as per figure 4.7 below.



```
  float activityMagnitude = calculateActivityMagnitude(incomingData.accelX, incomingData.accelY, incomingData.accelZ);
  int temperatureStatus = calculateTemperatureStatus(incomingData.temperature);
  int activityStatus = calculateActivityStatus(activityMagnitude);


  // Prepare input for RandomForest model
  float features[7] = {
    incomingData.accelX,
    incomingData.accelY,
    incomingData.accelZ,
    incomingData.temperature,
    (float)temperatureStatus,
    activityMagnitude,
    (float)activityStatus
  };

  // Predict health status using the model
  int prediction = model.predict(features);
  String state = (incomingData.temperature >= 27 && incomingData.temperature <= 34) ? "Normal" : "Attention Required";
```

```
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM8')
onnected to Wi-Fi
onnecting to AWS IoT Core...Connected to AWS IoT Core!

-- Data Received ---
ode ID: 2
emperature: 29.49
cceleration X: -0.76
cceleration Y: 0.44
cceleration Z: 9.69
ctivity Magnitude: 9.73
```

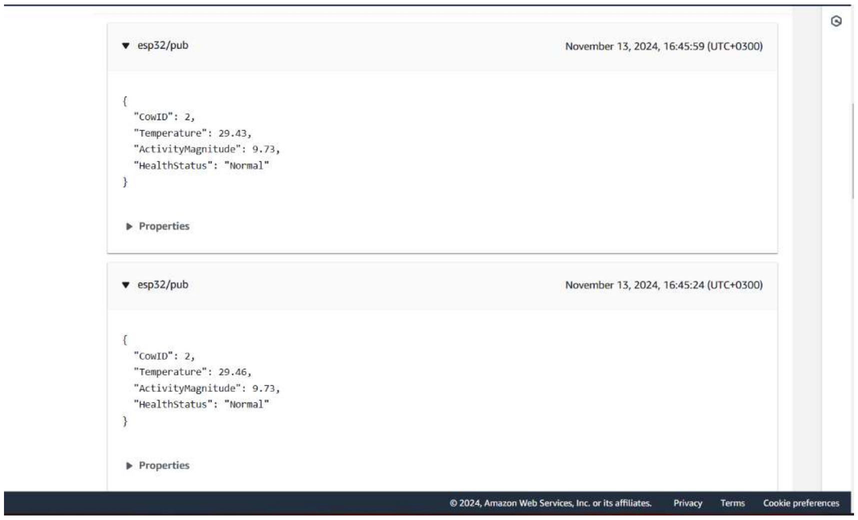
Figure 0.5 Data successfully received from the nodes



```
  while (!client.connected()) {
    Serial.print("Connecting to AWS IoT...");
    if (client.connect("ESP32Client")) { // Client ID can be any unique string
      Serial.println("connected!");
      client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
void loop() {
  if (!client.connected()) {
    setupAWSConnection();
  }
  client.loop();
}
```

```
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM8')
Connecting to WiFi...
Connecting to WiFi...
Connecting to WiFi...
Connected to WiFi
Connecting to AWS IoT...connected!
Message arrived on topic: esp32/sub. Message: {
  "message": "Hello from AWS IoT console"
}
```

Figure 0.6 Connection to AWS IoT Core



esp32/pub November 13, 2024, 16:45:59 (UTC+0300)

```
{
  "CowID": 2,
  "Temperature": 29.43,
  "ActivityMagnitude": 9.73,
  "HealthStatus": "Normal"
}
```

▶ Properties

esp32/pub November 13, 2024, 16:45:24 (UTC+0300)

```
{
  "CowID": 2,
  "Temperature": 29.46,
  "ActivityMagnitude": 9.73,
  "HealthStatus": "Normal"
}
```

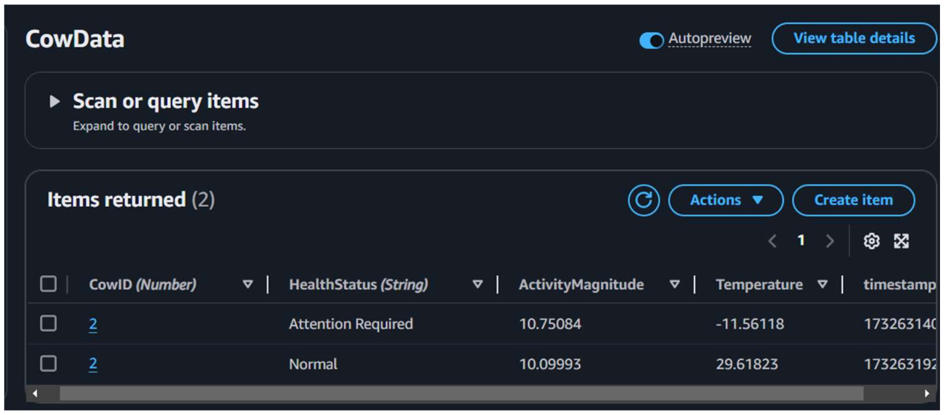
▶ Properties

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Figure 0.7 Processed data published to AWS IoT Core

4.2.3 Fetching of data from cloud database to mobile application

The processed data was received on the cloud platform are previously seen. However, the MQTT Test Client only holds data temporarily thus necessitating the use of a database to hold all the incoming data as per their node IDs. The rule that was created was able to save the data as shown in the figure below in DynamoDB which was then accessible by the mobile app. The figure 4.8 was a representation of how the data appeared in the DynamoDB.



CowID (Number)	HealthStatus (String)	ActivityMagnitude	Temperature	timestamp
2	Attention Required	10.75084	-11.56118	17326314
2	Normal	10.09993	29.61823	17326319

Figure 0.8 Data saved in DynamoDB

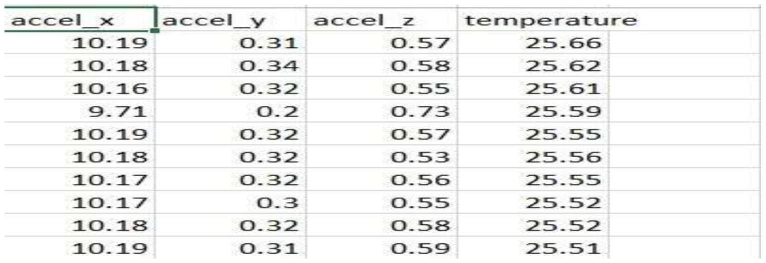
4.3 To Build and Deploy Machine Learning Models

The machine learning model demonstrated success in detecting anomalies and supporting real-time livestock health monitoring.

4.3.1 Data Collection and Preparation

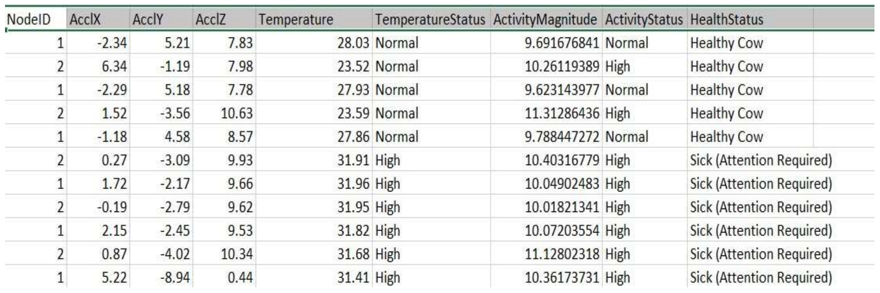
The automated data collection system reliably captured motion and temperature data under diverse conditions. Threshold-based labeling effectively distinguished between normal and anomalous states, providing a solid foundation for model training. Table 4.1 and 4.2 below show the data collected. This data was used t train and test the working of the machine learning model.

Table 0:1 Sample Data



accel_x	accel_y	accel_z	temperature
10.19	0.31	0.57	25.66
10.18	0.34	0.58	25.62
10.16	0.32	0.55	25.61
9.71	0.2	0.73	25.59
10.19	0.32	0.57	25.55
10.18	0.32	0.53	25.56
10.17	0.32	0.56	25.55
10.17	0.3	0.55	25.52
10.18	0.32	0.58	25.52
10.19	0.31	0.59	25.51

Table 0:2 Labelled data



NodeID	AcclX	AcclY	AcclZ	Temperature	TemperatureStatus	ActivityMagnitude	ActivityStatus	HealthStatus
1	-2.34	5.21	7.83	28.03	Normal	9.691676841	Normal	Healthy Cow
2	6.34	-1.19	7.98	23.52	Normal	10.26119389	High	Healthy Cow
1	-2.29	5.18	7.78	27.93	Normal	9.623143977	Normal	Healthy Cow
2	1.52	-3.56	10.63	23.59	Normal	11.31286436	High	Healthy Cow
1	-1.18	4.58	8.57	27.86	Normal	9.788447272	Normal	Healthy Cow
2	0.27	-3.09	9.93	31.91	High	10.40316779	High	Sick (Attention Required)
1	1.72	-2.17	9.66	31.96	High	10.04902483	High	Sick (Attention Required)
2	-0.19	-2.79	9.62	31.95	High	10.01821341	High	Sick (Attention Required)
1	2.15	-2.45	9.53	31.82	High	10.07203554	High	Sick (Attention Required)
2	0.87	-4.02	10.34	31.68	High	11.12802318	High	Sick (Attention Required)
1	5.22	-8.94	0.44	31.41	High	10.36173731	High	Sick (Attention Required)

4.3.2 Model Training Results

The Random Forest Classifier model achieved the following a model accuracy of 92% on the test set, confirming its robustness in identifying health anomalies. Figure 4.10 showed the machine learning results with the predicted clusters.

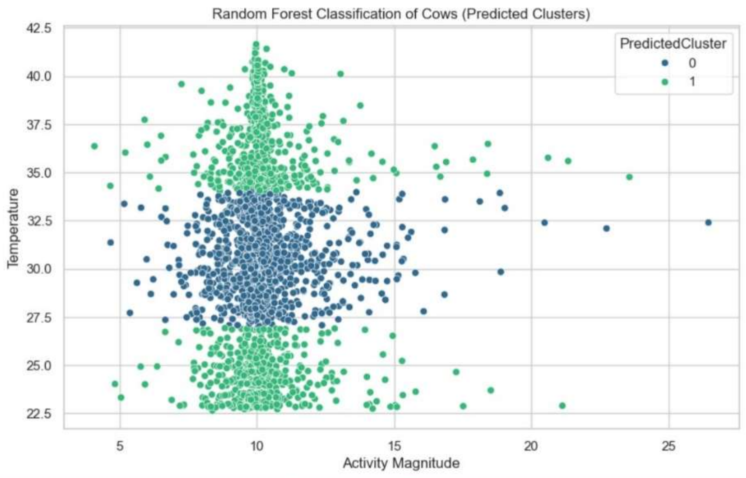


Figure 0.9 Machine learning model results

Feature Importance

Temperature and Activity Magnitude were identified as key features contributing to accurate anomaly detection. They were represented by the figure 4.11 below. The figures 4.12, 4.13, 4.14 and 4.15 showed the activity magnitude and temperature of the two cows as displayed by the machine learning model.

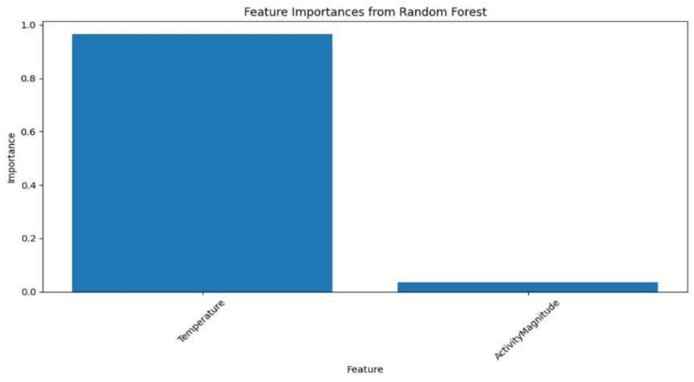


Figure 0.10 Feature importance from Random Forest

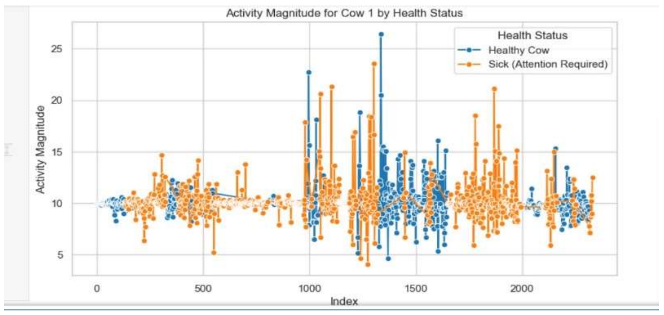


Figure 0.11 Activity magnitude for Cow 1

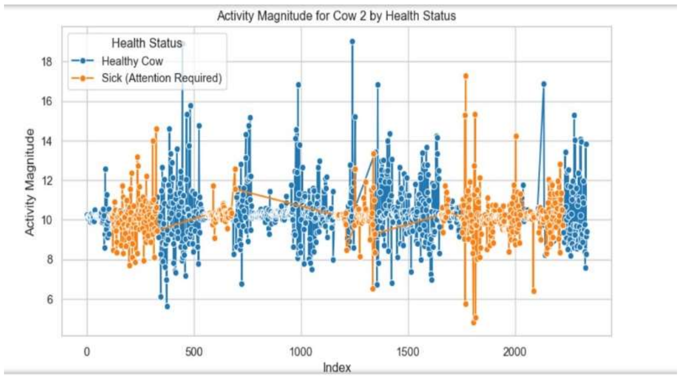


Figure 0.12 Activity magnitude for Cow 2

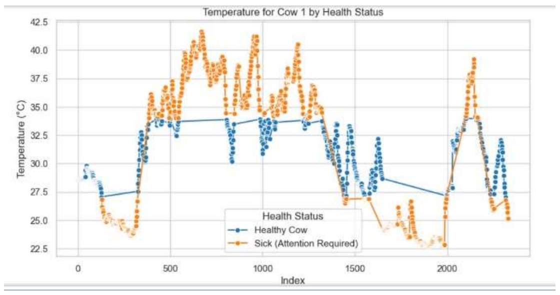


Figure 0.13 Temperature for Cow 1

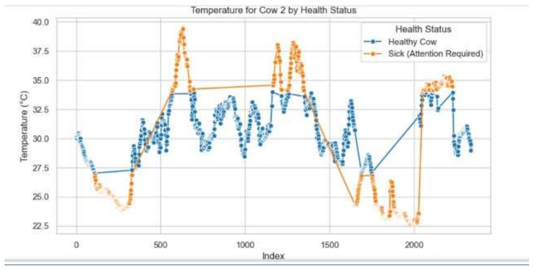
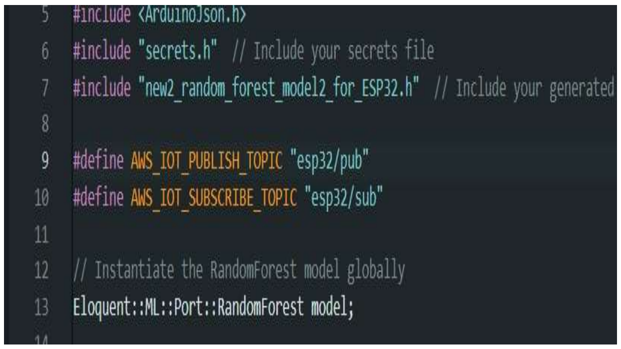


Figure 0.14 Temperature for Cow 2

4.3.3 Deployment and Real-Time Analysis

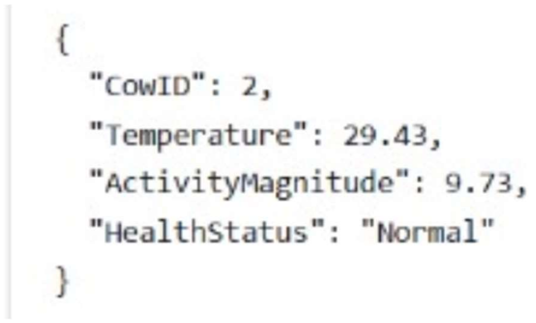
The deployed model enabled real-time anomaly detection at the ESP32 gateway. Key observations included effective anomaly identification, ensuring timely alerts for farmers, minimal false positives and enhancing the reliability of the notification system.

Figure 4.16 was a section of the code responsible for the machine learning model deployment on the gateway. Figures 4.17 and 4.18 were a representation of analyzed data results for a healthy and a sick cow respectively.



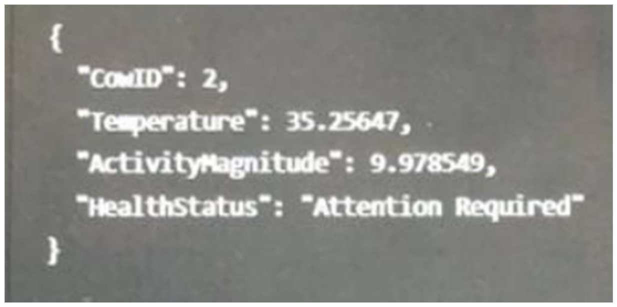
```
#include <ArduinoJson.h>
#include "secrets.h"  // Include your secrets file
#include "new2_random_forest_model2_for_ESP32.h"  // Include your generated

#define AWS_IOT_PUBLISH_TOPIC "esp32/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "esp32/sub"

// Instantiate the RandomForest model globally
Eloquent::ML::Port::RandomForest model;
```

Figure 0.15 ML Model deployment on gateway



```
{
  "CowID": 2,
  "Temperature": 29.43,
  "ActivityMagnitude": 9.73,
  "HealthStatus": "Normal"
}
```

Figure 0.16 Analyzed data results for healthy cow



```
{
  "CowID": 2,
  "Temperature": 35.25647,
  "ActivityMagnitude": 9.978549,
  "HealthStatus": "Attention Required"
}
```

Figure 0.17 Analyzed data results for abnormal cow

4.4 Mobile Application Development

The developed mobile application displayed real-time animal metrics. Farmers could create or modify their profiles, while maintaining the privacy of only viewing their animals. The dashboard was easy to use, and gave room for future expansion. Figure 4.18 captured the log in page, where the farmer used their email and password to log in. New users could also create new accounts through the create account tab.

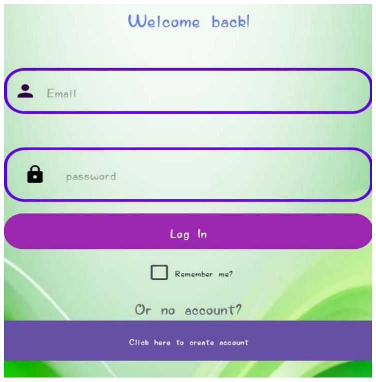


Figure 0.18 Mobile app log in page

Figure 4.19 was the page where new farmers created new accounts.

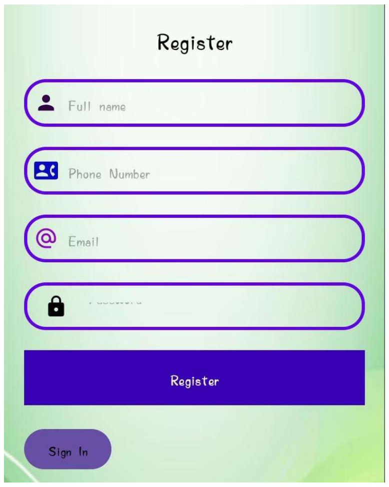


Figure 0.19 Create Account Page

Figure 4.20 showed the dashboard page, which would redirect the farmers to the animals tab. It also prompted the farmer to log out or adjust in-app settings.

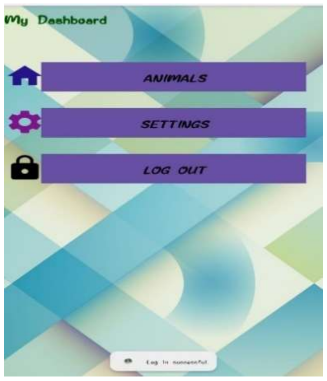


Figure 0.20 Dashboard Page

Figure 4.21 fetched the animals by the names displayed from the machine learning model. They appeared by the order of names as listed in the machine learning model.

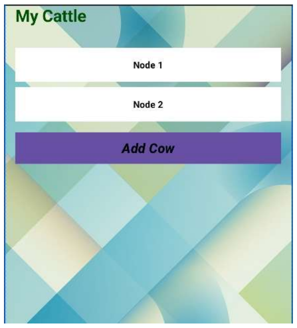


Figure 0.21 The animals page as displayed in the dynamoDB

Figure 4.21 was used to fetch animals from the model. It separated the animals, giving room for future expansions.

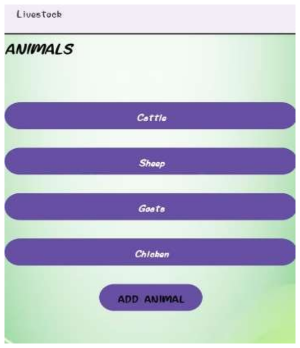


Figure 0.22 Page showing different animals for future expansion

Figure 4.23 and figure 4.24 were the cows' specific metrics, as fetched from the dynamoDB. They represented real-time data, as it keyed-in the dynamoDB.

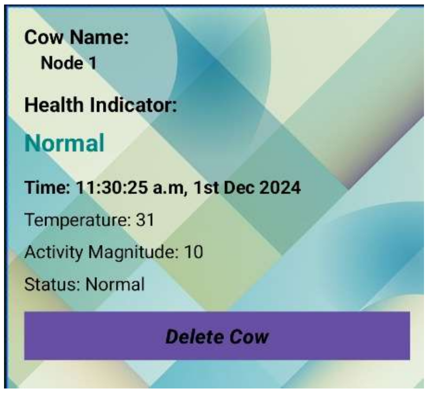

Cow Name:
Node 1
Health Indicator:
Normal
Time: 11:30:25 a.m, 1st Dec 2024
Temperature: 31
Activity Magnitude: 10
Status: Normal

Figure 0.23 Real-time metrics for Cow 1

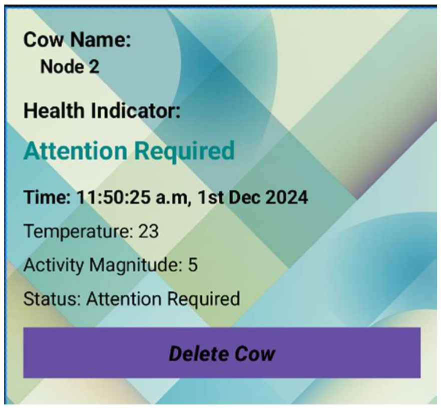

Cow Name:
Node 2
Health Indicator:
Attention Required
Time: 11:50:25 a.m, 1st Dec 2024
Temperature: 23
Activity Magnitude: 5
Status: Attention Required

Figure 0.24 Real-time metrics for cow 2

CHAPTER 5: CONCLUSION AND RECOMMENDATION

CONCLUSION

The Livestock Health and Activity Monitoring System using IoT-based sensor technology successfully met the outlined objectives and achieved the project’s aim. The design and development of wearable IoT devices were accomplished with compact and reliable devices that integrated MPU6050 (IMU) and temperature sensors. Reliable data transmission and processing were achieved through seamless communication from sensor nodes to the central gateway and AWS cloud, where real-time data analysis was performed. The Random Forest Classifier machine learning model was developed and deployed with an impressive accuracy of 92%, successfully detecting health anomalies in livestock based on movement and temperature data. The project also delivered a user-friendly mobile application, which provided farmers with real-time insights into their livestock's health, enabling them to respond promptly to any detected issues. The system achieved significant improvements in accuracy, with motion tracking at 95% and temperature deviations of only ±0.2°C, compared to traditional monitoring methods. Furthermore, efficiency was enhanced through real-time anomaly detection, allowing for timely interventions and reducing veterinary costs. Overall, the system provides a scalable, reliable, and cost-effective solution for improving livestock health and productivity.

RECOMMENDATION

While the project achieved its goals, future improvements can further enhance its performance. Sensor placement optimization should be explored to determine the ideal positioning for accurately monitoring cattle behaviors such as lying or rumination. Integrating advanced sensors to monitor additional physiological parameters, such as heart rate and respiration, would allow for the detection of a broader range of health conditions. Battery life can be extended by incorporating energy-efficient microcontrollers or solar-powered IoT nodes, ensuring prolonged operation in remote field conditions. To improve scalability, the system can adopt advanced multi-node communication protocols to handle larger cattle herds effectively. Future researchers could also consider using a long-range transmission protocol to handle hers in a large geographical area. Additionally, future research could focus on enhancing the machine learning models by incorporating larger and more diverse datasets, as well as including deep learning techniques to identify complex health patterns. By addressing these areas, the system can be improved to provide an even more comprehensive, efficient, and intelligent livestock health monitoring solution.

REFERENCES

[1] M. N. Lukuyu et al., "Farmers' Perceptions of Dairy Cattle Breeds, Breeding and Feeding Strategies: A Case of Smallholder Dairy Farmers in Western Kenya," *East Afr. Agric. For. J.*, vol. 83, no. 4, pp. 351–367, Dec. 2019, doi: 10.1080/00128325.2019.1659215.

[2] M. Dayoub et al., "Enhancing Animal Production through Smart Agriculture: Possibilities, Hurdles, Resolutions, and Advantages," *Ruminants*, vol. 4, no. 1, pp. 22–46, Mar. 2024, doi: https://doi.org/10.3390/ruminants4010003.

[3] C. M. Mburu et al., "Prioritization of livestock diseases by pastoralists in Oloitoktok Sub County, Kajiado County, Kenya," *PLOS ONE*, vol. 18, no. 7, Jul. 2023, doi: 10.1371/journal.pone.0287456.

[4] E. C. Chepkwony et al., "Epidemiological study on foot-and-mouth disease in small ruminants: Sero-prevalence and risk factor assessment in Kenya," *PLOS ONE*, vol. 16, no. 8, Aug. 2021, doi: 10.1371/journal.pone.0234286.

[5] M. Halachmi, M. Guarino, J. Bewley, and M. Pastell, "Smart Animal Agriculture: Application of Real-Time Sensors to Improve Animal Well-Being and Production," *Annual Review of Animal Biosciences*, vol. 7, no. 1, pp. 403–425, Feb. 2019, doi: https://doi.org/10.1146/annurev-animal-020518-114851.

[6] Andriamandroso, A.L.H.; Bindelle, J.; Mercatoris, B.; Lebeau, F. A review on using sensors to monitor cattle jaw movements and grazing behavior. Biotechnol. Agron. Soc. Environ. 2016, 20, 273–286.

[7] Benaissa, S.; Tuyttens, F.A.M.; Plets, D.; Trogh, J.; Martens, L.; Vandaele, L.; Joseph, W.; Sonck, B. Calving and estrus detection in dairy cattle using a combination of indoor localization and accelerometer sensors. Comput. Electron. Agric. 2020, 168, 105153.

[8] Singh A. Common cattle diseases: Symptoms, treatment and prevention. InDairy Year Book 2014 (pp. 446-453). Coll. Vet. Sci. Anim. Husbandry Mhow.

[9] Niu L, Yang C, Du Y, Qin L, Li B. Cattle disease auxiliary diagnosis and treatment system based on data analysis and mining. In2020 5th International Conference on Computer and Communication Systems (ICS) 2020 May 15 (pp. 24-27). IEEE.

[10] Wang, J.; He, Z.; Ji, J.; Zhao, K.; Zhang, H. IoT-based measurement system for classifying cow behavior from tri-axial accelerometer. Ciência Rural 2019, 49

[11] Gloster J, Ebert K, Gubbins S, Bashiruddin J, Paton DJ. Normal variation in thermal radiated temperature in cattle: implications for foot-and-mouth disease detection. BMC veterinary research. 2011 Dec;7:1-0.

[12] Zhang, M.; Wang, X.; Feng, H.; Huang, Q.; Xiao, X.; Zhang, X. Wearable Internet of Things enabled precision livestock farming in smart farms: A review of technical solutions for precise perception, biocompatibility, and sustainability monitoring. J. Clean. Prod. 2021, 312, 127712.

[13] Stangaferro, M.L.; Wijma, R.; Caixeta, L.S.; Al-Abri, M.A.; Giordano, J.O. Use of rumination and activity monitoring for the identification of dairy cows with health disorders: Part I. Metabolic and digestive disorders. J. Dairy Sci. 2016, 99, 7395–7410.

[14] Henchion, M.M.; Regan, Á.; Beecher, M.; MackenWalsh, Á. Developing 'Smart' Dairy Farming Responsive to Farmers and Consumer-Citizens: A Review. Animals 2022, 12, 360.

[15] Van Nuffel, A.; Zwertvaegher, I.; Van Weyenberg, S.; Pastell, M.; Thorup, V.; Bahr, C.; Sonck, B.; Saeys, W. Lameness Detection in Dairy Cows: Part 2. Use of Sensors to Automatically Register Changes in Locomotion or Behavior. Animals 2015, 5, 861–885.

[16] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 2015, arXiv:1409.1556.

[17] McManus, C.; Tanure, C.B.; Peripolli, V.; Seixas, L.; Fischer, V.; Gabbi, A.M.; Menegassi, S.R.O.; Stumpf, M.T.; Kolling, G.J.; Dias, E.; et al. Infrared thermography in animal production: An overview. Comput. Electron. Agric. 2016, 123, 10–16.

[18] Alsaaod, M.; Schaefer, A.L.; Büscher, W.; Steiner, A. The Role of Infrared Thermography as a Non-Invasive Tool for the Detection of Lameness in Cattle. Sensors 2015, 15, 14513–14525.

[19] Nasirahmadi, A., et al. "Behavioral monitoring of pigs and cattle using machine vision and machine learning techniques." Computers and Electronics in Agriculture, vol. 167, 2019, pp. 105041.

[20] Maltz, E., et al. "Towards wireless communication in precision dairy farming." Computers and Electronics in Agriculture, vol. 64, no. 1, 2008, pp. 59-69

[21] Guo, L., et al. "Automatic detection of lameness in dairy cattle using a pressure-sensitive walkway and machine learning algorithms." Journal of Dairy Science, vol. 101, no. 7, 2018, pp. 6323-6335

APPENDIX

Appendix 1: Budget

Table 6:1 Project Budget

ITEM	DESCRIPTION	PRICE PER ITEM	QUANTITY	TOTAL
DEVKITV1 WROOM	Micro-controller IC by ESP32	1000	3	3000
PCB BOARD	Etched PCB with schematic design	100	1	100
MPU 6050	IMU sensor with accelerometer, gyroscope and temperature sensor	500	2	1000
3.7v Lithium battery	Rechargeable Batteries	200	2	400
Miscellaneous	Connectors, power regulator	500	1	500
Grand total				5000

Appendix 2: Code used in the project

Node

```
#include <Adafruit_MPU6050.h>

#include <Adafruit_Sensor.h>

#include <esp_now.h>

#include <WiFi.h>

// Initialize MPU6050

Adafruit_MPU6050 mpu;

// Define the MAC address of the gateway

uint8_t gatewayAddress[] = {0xEC, 0x64, 0xC9, 0x85, 0x61, 0x98}; // Update with your gateway's MAC address

// Define a unique node ID for this sensor node

const uint8_t NODE_ID = 1;

#include <Adafruit_MPU6050.h>

#include <Adafruit_Sensor.h>

#include <esp_now.h>

#include <WiFi.h>

// Initialize MPU6050

Adafruit_MPU6050 mpu;

// Define the MAC address of the gateway

uint8_t gatewayAddress[] = {0xEC, 0x64, 0xC9, 0x85, 0x61, 0x98}; // Update with your gateway's MAC address

// Define a unique node ID for this sensor node

const uint8_t NODE_ID = 1;

// Structure to hold sensor data

typedef struct struct_message {

  uint8_t nodeID;        // Unique ID for each node

  float temperature;     // Temperature from MPU6050

  float accelX;

  float accelY;

  float accelZ;

} struct_message;

struct_message sensorData;

void setup() {

  Serial.begin(115200);

  // Initialize MPU6050

  if (!mpu.begin()) {

    Serial.println("Failed to find MPU6050 chip");

    while (1) {

      delay(10);

    } }

mpu.setAccelerometerRange(MPU6050_RANGE_2_G);

mpu.setGyroRange(MPU6050_RANGE_250_DEG);

mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  Serial.println("MPU6050 initialized");

  // Initialize WiFi in station mode

  WiFi.mode(WIFI_STA);

  // Initialize ESP-NOW

  if (esp_now_init() != ESP_OK) {

    Serial.println("Error initializing ESP-NOW");

    return;

  }

  // Register the gateway's MAC address

  esp_now_peer_info_t peerInfo;

  memcpy(peerInfo.peer_addr, gatewayAddress, 6);

  peerInfo.channel = 0;

  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK) {

    Serial.println("Failed to add peer");

    return;

  }}

void loop() {

  // Read accelerometer and temperature data from MPU6050

  sensors_event_t a, g, temp;

  mpu.getEvent(&a, &g, &temp);

  sensorData.temperature = temp.temperature;  // Get temperature from MPU6050

  // Read accelerometer data from MPU6050

  sensorData.accelX = a.acceleration.x;

  sensorData.accelY = a.acceleration.y;

  sensorData.accelZ = a.acceleration.z;

  // Add node ID to sensor data

  sensorData.nodeID = NODE_ID;

  // Print the sensor data to the Serial Monitor

  Serial.print("Node ID: ");

  Serial.println(sensorData.nodeID);

  Serial.print("Temperature: ");

Serial.println(sensorData.temperature);

  Serial.print("AccelX: ");

  Serial.println(sensorData.accelX);

  Serial.print("AccelY: ");

  Serial.println(sensorData.accelY);

  Serial.print("AccelZ: ");

  Serial.println(sensorData.accelZ);

  Serial.println();  // Print a blank line between each data set

  // Send data via ESP-NOW
  esp_err_t result = esp_now_send(gatewayAddress, (uint8_t *)&sensorData, sizeof(sensorData));

  if (result == ESP_OK) {

    Serial.println("Data sent successfully");

  } else {

    Serial.println("Error sending data"); }

  delay(5000);  // Delay between sends

}
```

Gateway

```
#include <esp_now.h>

#include <esp_wifi.h>

#include <WiFi.h>

#include <WiFiClientSecure.h>

#include <PubSubClient.h>

#include <ArduinoJson.h>

#include "secrets.h"  // Include your secrets file

#include "new2_random_forest_model2_for_ESP32.h"  // Include your generated C model

#define ESP_MAC_WIFI_STA 0

#define AWS_IOT_PUBLISH_TOPIC "esp32/pub"

#define AWS_IOT_SUBSCRIBE_TOPIC "esp32/sub"

// Wi-Fi credentials

#define WIFI_SSID "QUOVADIS YOUTH HUB"

#define WIFI_PASSWORD "Quo@2023"

// Instantiate the RandomForest model globally

Eloquent::ML::Port::RandomForest model;

// MQTT client and secure Wi-Fi client

WiFiClientSecure net;

PubSubClient client(net);

// Structure to hold incoming sensor data

typedef struct SensorData {

  uint8_t nodeID;

  float temperature;

  float accelX;

  float accelY;

  float accelZ;

} SensorData;

// Helper functions

int calculateTemperatureStatus(float temperature) {

  if (temperature > 34) return 1;  // Elevated temperature

  else if (temperature < 27) return -1;  // Low temperature

  return 0;  // Normal temperature

}

float calculateActivityMagnitude(float accelX, float accelY, float accelZ) {

  return sqrt(accelX * accelX + accelY * accelY + accelZ * accelZ);

}

int calculateActivityStatus(float activityMagnitude) {

  if (activityMagnitude > 10) return 1;  // High activity

  else if (activityMagnitude < 7) return -1;  // Low activity

  return 0;  // Normal activity

}

// Callback function for receiving data via ESP-NOW

void onDataReceive(const esp_now_recv_info_t *info, const uint8_t *data, int len) {

  Serial.println("onDataReceive triggered");

  Serial.print("Data length received: ");

Serial.println(len);

Serial.print("Expected length: ");

Serial.println(sizeof(SensorData));

  // Ensure we don't read more data than expected

  //if (len != sizeof(SensorData)) {

   //Serial.println("Received incorrect data length");

   //return;

  //}

   // Copy the incoming data into the structure

  SensorData incomingData;

  memcpy(&incomingData, data, sizeof(incomingData));

  // Calculate additional metrics

  float activityMagnitude = calculateActivityMagnitude(incomingData.accelX, incomingData.accelY, incomingData.accelZ);

  int temperatureStatus = calculateTemperatureStatus(incomingData.temperature);

  int activityStatus = calculateActivityStatus(activityMagnitude);

  // Prepare input for RandomForest model

  float features[7] = {

    incomingData.accelX,

    incomingData.accelY,

    incomingData.accelZ,

    incomingData.temperature,

    (float)temperatureStatus,

    activityMagnitude,

    (float)activityStatus

  };

  // Predict health status using the model

  int prediction = model.predict(features);
  String state = (incomingData.temperature >= 27 && incomingData.temperature <= 34) ? "Normal" : "Attention Required";

  //Prepare JSON payload for AWS IoT Core

  DynamicJsonDocument doc(256);

  doc["CowID"] = incomingData.nodeID;

  doc["Temperature"] = incomingData.temperature;

  doc["ActivityMagnitude"] = activityMagnitude;

  doc["HealthStatus"] = state;

  String json;

  serializeJson(doc, json);



  // Publish to AWS IoT Core

  if (!client.publish(AWS_IOT_PUBLISH_TOPIC, json.c_str())) {

    Serial.println("Failed to publish message");

  } else {

    Serial.println("Message successfully published to AWS IoT Core");

  }}

void setup() {

  Serial.begin(115200);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  while (WiFi.status() != WL_CONNECTED) {

    delay(1000);

    Serial.println("Connecting to WiFi...");

  }

  Serial.println("Connected to WiFi");

  // Print the MAC address

  Serial.print("MAC Address: ");

  Serial.println(WiFi.macAddress());

  setupAWSConnection();

 // Step 2: Check ESP-NOW initialization

  if (esp_now_init() == ESP_OK) {

    Serial.println("ESP-NOW Initialized successfully");

  } else {

    Serial.println("ESP-NOW Initialization failed");

    return;  // Stop setup if ESP-NOW initialization failed

  }

  // Step 3: Register receive callback

  if (esp_now_register_recv_cb(onDataReceive) == ESP_OK) {

    Serial.println("Callback registered successfully");

  } else {

    Serial.println("Failed to register callback");

    return;  // Stop setup if callback registration failed

  }

   // Step 4: Print MAC address to ensure pairing is correct

  //uint8_t mac[6];

  //esp_read_mac(mac, ESP_MAC_WIFI_STA);

  //Serial.printf("MAC Address: %02X:%02X:%02X:%02X:%02X:%02X\n", mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);

  //Serial.println("Setup complete");

}

// Helper functions (same as in your code)

void mqttCallback(char* topic, byte* payload, unsigned int length) {

  Serial.print("Message arrived on topic: ");

  Serial.print(topic);

  Serial.print(". Message: ");

  for (int i = 0; i < length; i++) {

    Serial.print((char)payload[i]);

  }

  Serial.println();

}

void setupAWSConnection() {

  net.setCACert(AWS_ROOT_CA);

  net.setCertificate(CERTIFICATE);

  net.setPrivateKey(PRIVATE_KEY);

client.setServer(AWS_IOT_ENDPOINT, 8883);

  client.setCallback(mqttCallback);

  while (!client.connected()) {

    Serial.print("Connecting to AWS IoT...");

    if (client.connect("ESP32Client")) { // Client ID can be any unique string

      Serial.println("connected!");

client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);

    } else {

      Serial.print("failed, rc=");

      Serial.print(client.state());

      Serial.println(" try again in 5 seconds");

      delay(5000); }}}



void loop() {

  SensorData incomingData;

  float activityMagnitude = calculateActivityMagnitude(incomingData.accelX, incomingData.accelY, incomingData.accelZ);

  String state = (incomingData.temperature >= 27 && incomingData.temperature <= 34) ? "Normal" : "Attention Required";

  if (WiFi.status() == WL_CONNECTED) {
```

```
  // Prepare JSON payload

  String jsonData = "{\"CowID\": 1, \"Temperature\": 29.5, \"ActivityMagnitude\": 0.75, \"HealthStatus\": \"Normal\"}";

   // Print the received data to the Serial Monitor

  Serial.println("\n--- Data Received ---");

  Serial.print("Node ID: ");

Serial.println(incomingData.nodeID);

  Serial.print("Temperature: ");

Serial.println(incomingData.temperature);

  Serial.print("Acceleration X: ");

Serial.println(incomingData.accelX);

  Serial.print("Acceleration Y: ");

Serial.println(incomingData.accelY);

  Serial.print("Acceleration Z: ");

  Serial.println(incomingData.accelZ);

  Serial.print("Activity Magnitude: ");

  Serial.println(activityMagnitude);

  // Print health status

  Serial.print("Health Status (Prediction): ");

  Serial.println(state);

  }

 if (!client.connected()) {

  setupAWSConnection();

 }

 client.loop();

 delay(10000);  // Wait 10 seconds before sending next data

}
```

Data Saving Automation Script

```
import serial

import csv

import time

# Initialize serial connection (update COM port and baud rate as needed)

ser = serial.Serial('COM8', 115200)  # Replace 'COM8' with your port (e.g., '/dev/ttyUSB0' on Linux/Mac)

# Open a CSV file to store data

with open('livestock_data.csv', 'w', newline='', encoding="utf-8") as file:

    writer = csv.writer(file)

    # Write the header

    writer.writerow(["Timestamp", "accelX", "accelY", "accelZ", "Temperature"])

        while True:

        try:

            # Read a line from the serial port

            line = ser.readline().decode('utf-8', errors='replace').strip()

            data = line.split('\t')  # Assuming data is tab-separated

            # Get the current timestamp

            timestamp = time.strftime('%Y-%m-%d %H:%M:%S')

            # Write the data to the CSV file

            writer.writerow([timestamp] + data)

            file.flush()   # Ensure data is written to disk after each row

            print(f"Data logged: {timestamp}, {data}")

        except KeyboardInterrupt:

            print("Data collection stopped.")

            break
```

Random Forest Classifier

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

import joblib

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

import os

# Read the CSV File

file_path = r'C:\Users\Courtney\Desktop\5.2\sensor_data.csv'  # Prefix with 'r'

data = pd.read_csv(file_path)

# Define the correct file paths

input_file_path = r'C:\Users\Courtney\Desktop\5.2\sensor_data.csv'

output_file_path = r'C:\Users\Courtney\Desktop\5.2\labeledsensor_data_file.csv'

# Function to classify temperature

def classify_temperature(temp):

    if temp < 27:

        return 'Low'

    elif temp > 34:

        return 'High'

    else:

        return 'Normal'

# Calculate the magnitude of activity from acceleration data

def calculate_activity_magnitude(row):

    return np.sqrt(row['AcclX']**2 + row['AcclY']**2 + row['AcclZ']**2)

# Define thresholds for activity

def categorize_activity(activity_magnitude):

    if activity_magnitude < 7:

        return 'Low'
    elif activity_magnitude > 10:

        return 'High'

    else:

        return 'Normal'

# Simplified function to label health status with default state as "Sick (Attention Required)"

def label_health_status(row):

    temp_status = classify_temperature(row['Temperature'])

    activity_magnitude = calculate_activity_magnitude(row)

    activity_status = categorize_activity(activity_magnitude)

        if temp_status == 'Normal' and activity_status in ['Low','Normal','High']:

        return 'Healthy Cow'

    else:

        return 'Sick (Attention Required)'

# Check if the input file exists

if os.path.isfile(input_file_path):

    try:

        # Load the CSV file into a DataFrame

        df = pd.read_csv(input_file_path)

        # Apply the functions to classify and label

        df['TemperatureStatus'] = df['Temperature'].apply(classify_temperature)

        df['ActivityMagnitude'] = df.apply(calculate_activity_magnitude, axis=1)

        df['ActivityStatus'] = df['ActivityMagnitude'].apply(categorize_activity)

        df['HealthStatus'] = df.apply(label_health_status, axis=1)

        # Save the updated DataFrame to a new CSV file

        df.to_csv(output_file_path, index=False

        # Verify if the file was created

        if os.path.isfile(output_file_path):

            print(f"File successfully saved to: {output_file_path}")

        else:

            print(f"File not found at the expected path: {output_file_path}")

    except pd.errors.EmptyDataError:

        print("The file is empty.")

    except pd.errors.ParserError:

        print("Error parsing the file. It may not be a valid CSV.")

    except Exception as e:

        print(f"An unexpected error occurred: {e}")

else:

    print(f"Input file not found at the path: {input_file_path}")

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier

import joblib

# Load the dataset (update with the path to your CSV file)

file_path = r'C:\Users\Courtney\Desktop\5.2\labeledsensor_data_file.csv'

data = pd.read_csv(file_path)

# Convert categorical columns to numerical using Label Encoding

le = LabelEncoder()

# Encode the 'HealthStatus' column

data['HealthStatus'] = le.fit_transform(data['HealthStatus'])  # Target variable

# Select features (input variables) and the target variable

# Only include features used in the Arduino code

X = data[['Temperature', 'ActivityMagnitude']]

y = data['HealthStatus']

# Split the dataset into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)

# Output the shapes of training and testing data

print(f"Training data shape: {X_train.shape}, Test data shape: {X_test.shape}")

print(f"Training labels shape: {y_train.shape}, Test labels shape: {y_test.shape}")

# Train the RandomForest model

model = RandomForestClassifier()

model.fit(X_train, y_train)

# Save the trained model

joblib.dump(model, 'random_forest_model.pkl')

from sklearn.preprocessing import LabelEncoder

import pandas as pd

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, make_scorer

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

import joblib

from micromlgen import port
```

```
# Load the dataset (update with the path to your CSV file)

file_path = r'C:\Users\Courtney\Desktop\5.2\labeledsensor_data_file.csv'

data = pd.read_csv(file_path)

# Encode the target variable

le = LabelEncoder()

data['HealthStatus'] = le.fit_transform(data['HealthStatus']) # Converts 'Healthy Cow' and 'Sick (Attention Required)' to numeric

# Select only the features you are using in the ESP32 code: Temperature and ActivityMagnitude

X = data[['Temperature', 'ActivityMagnitude']]

y = data['HealthStatus']

# Define preprocessor to scale numerical features (in this case, only 2 features)

preprocessor = ColumnTransformer(

    transformers=[

        ('num', StandardScaler(), ['Temperature', 'ActivityMagnitude'])

    ])

# Create a pipeline that preprocesses the data and then applies the Random Forest model

pipeline = Pipeline(steps=[

    ('preprocessor', preprocessor),

    ('classifier', RandomForestClassifier(n_estimators=100, random_state=100))

])

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)



# Train the model

pipeline.fit(X_train, y_train)

# Make predictions on the test data

y_pred = pipeline.predict(X_test)

# Define the positive label after encoding (assuming 'Sick (Attention Required)' was encoded as 1)

positive_label = 1

# Evaluate the model's performance on the test data

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred, pos_label=positive_label)

recall = recall_score(y_test, y_pred, pos_label=positive_label)

f1 = f1_score(y_test, y_pred, pos_label=positive_label)

# Print the results

print("Test Data Performance:")

print(f"Accuracy: {accuracy * 100:.2f}%")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1-Score: {f1:.2f}")

# Define custom scorers with the appropriate positive label

precision_scorer = make_scorer(precision_score, pos_label=positive_label)

recall_scorer = make_scorer(recall_score, pos_label=positive_label)

f1_scorer = make_scorer(f1_score, pos_label=positive_label)

# Perform cross-validation with 5 folds

cv_accuracy = cross_val_score(pipeline, X, y, cv=5, scoring='accuracy')

cv_precision = cross_val_score(pipeline, X, y, cv=5, scoring=precision_scorer)

cv_recall = cross_val_score(pipeline, X, y, cv=5, scoring=recall_scorer)

cv_f1 = cross_val_score(pipeline, X, y, cv=5, scoring=f1_scorer)

# Print the average cross-validation results

print("\nCross-Validation Performance:")

print(f"Cross-Validation Accuracy: {cv_accuracy.mean() * 100:.2f}%")

print(f"Cross-Validation Precision: {cv_precision.mean():.2f}")

print(f"Cross-Validation Recall: {cv_recall.mean():.2f}")

print(f"Cross-Validation F1-Score: {cv_f1.mean():.2f}")

# After training the RandomForest model, save the model

joblib.dump(pipeline, 'new2_random_forest_model.pkl')

# Convert the model to C code using micromlgen for deployment on ESP32

c_code = port(pipeline.named_steps['classifier'])

# Save the generated C code to a file

with open('new2_random_forest_model2_for_ESP32.c', 'w') as f:

    f.write(c_code)

import matplotlib.pyplot as plt

import numpy as np

# Extract feature importances from the trained RandomForest model

importances = pipeline.named_steps['classifier'].feature_importances_

# Updated features list based on your current model setup

features = ['Temperature', 'ActivityMagnitude']

# Display the feature importances alongside their corresponding feature names

for feature, importance in zip(features, importances):

    print(f"Feature: {feature}, Importance: {importance:.4f}")

# Sort the feature importances in descending order

indices = np.argsort(importances)[::-1]

# Plot the feature importances
```

```
plt.figure(figsize=(10, 6))

plt.title("Feature Importances from Random Forest")

plt.bar(range(len(importances)), importances[indices], align="center")

plt.xticks(range(len(importances)), [features[i] for i in indices], rotation=45)

plt.ylabel("Importance")

plt.xlabel("Feature")

plt.tight_layout()



# Display the plot

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

# Assuming 'data' DataFrame contains ActivityMagnitude, Temperature, AcclX, AcclY, AcclZ, and 'HealthStatus'

# Define the feature columns

features = ['ActivityMagnitude', 'Temperature', 'AcclX', 'AcclY', 'AcclZ']

# Split the data into features (X) and target (y)

X = data[features]

y = data['HealthStatus']  # Assuming this column represents the health status (e.g., 'Healthy' or 'Sick')

# Split into training and test sets (optional, could use all data for visualization)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Random Forest Classifier

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

rf_classifier.fit(X_train, y_train)

# Predict the health status (or cluster) for the entire dataset

data['PredictedCluster'] = rf_classifier.predict(X)

# Plot the scatterplot for 'ActivityMagnitude' vs 'Temperature', colored by the predicted cluster

plt.figure(figsize=(10, 6))

sns.scatterplot(data=data, x='ActivityMagnitude', y='Temperature', hue='PredictedCluster', palette='viridis')

plt.title("Random Forest Classification of Cows (Predicted Clusters)")

plt.xlabel("Activity Magnitude")

plt.ylabel("Temperature")

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load your data

data = pd.read_csv(r'C:\Users\Courtney\Desktop\5.2\labeledsensor_data_file.csv')

# Filter out 'Unknown Status' from the dataset

data = data[data['HealthStatus'].isin(['Healthy Cow', 'Sick (Attention Required)'])]

# Set up the plotting style

sns.set(style="whitegrid")

# Separate the data by NodeID (Cows 1 and 2)

data_cow_1 = data[data['NodeID'] == 1]

data_cow_2 = data[data['NodeID'] == 2]

# Function to plot Activity Magnitude for each cow and compare health

def plot_activity_magnitude(data, cow_id):

    plt.figure(figsize=(10, 5))

    sns.lineplot(data=data, x=data.index, y='ActivityMagnitude', hue='HealthStatus', marker='o', palette='tab10')

    plt.title(f"Activity Magnitude for Cow {cow_id} by Health Status")

    plt.xlabel('Index')

    plt.ylabel('Activity Magnitude')

    plt.legend(title='Health Status')

    plt.grid(True)

    plt.show()

# Plot Activity Magnitude for Cow 1

plot_activity_magnitude(data_cow_1, cow_id=1)

# Plot Activity Magnitude for Cow 2

plot_activity_magnitude(data_cow_2, cow_id=2)

# Function to plot other parameters for comparison between cows

def plot_parameter(data, parameter, title, ylabel, cow_id):

    plt.figure(figsize=(10, 5))

    sns.lineplot(data=data, x=data.index, y=parameter, hue='HealthStatus', marker='o', palette='tab10')

    plt.title(f"{title} for Cow {cow_id} by Health Status")

    plt.xlabel('Index')

    plt.ylabel(ylabel)

    plt.legend(title='Health Status')

    plt.grid(True)

    plt.show()

# Plot Temperature for Cow 1

plot_parameter(data_cow_1, 'Temperature', 'Temperature', 'Temperature (°C)', cow_id=1)

# Plot Temperature for Cow 2

plot_parameter(data_cow_2, 'Temperature', 'Temperature', 'Temperature (°C)', cow_id=2)

# Plot Acceleration X for Cow 1
```

```
plot_parameter(data_cow_1, 'AcclX', 'Acceleration X', 'Acceleration X', cow_id=1)

# Plot Acceleration X for Cow 2

plot_parameter(data_cow_2, 'AcclX', 'Acceleration X', 'Acceleration X', cow_id=2)

# Plot Acceleration Y for Cow 1

plot_parameter(data_cow_1, 'AcclY', 'Acceleration Y', 'Acceleration Y', cow_id=1)

# Plot Acceleration Y for Cow 2

plot_parameter(data_cow_2, 'AcclY', 'Acceleration Y', 'Acceleration Y', cow_id=2)

# Plot Acceleration Z for Cow 1

plot_parameter(data_cow_1, 'AcclZ', 'Acceleration Z', 'Acceleration Z', cow_id=1)

# Plot Acceleration Z for Cow 2

plot_parameter(data_cow_2, 'AcclZ', 'Acceleration Z', 'Acceleration Z', cow_id=2)

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load your data

data = pd.read_csv(r'C:\Users\Courtney\Desktop\5.2\labeleddata_data_file.csv')

# Filter out 'Unknown Status' from the dataset

data = data[data['HealthStatus'].isin(['Healthy Cow', 'Sick (Attention Required)'])]

# Select only numeric columns for correlation matrix

numeric_data = data.select_dtypes(include=[float, int])

# Calculate the correlation matrix

correlation_matrix = numeric_data.corr()

# Plot the correlation matrix

plt.figure(figsize=(10, 8))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title("Correlation Matrix")

plt.show()

from sklearn.ensemble import IsolationForest

# Apply Isolation Forest for anomaly detection

model = IsolationForest(contamination=0.05, random_state=42)

data['Anomaly'] = model.fit_predict(data[['ActivityMagnitude', 'Temperature', 'AcclX', 'AcclY', 'AcclZ']])

# Visualize anomalies

sns.scatterplot(data=data, x='ActivityMagnitude', y='Temperature', hue='Anomaly', palette='coolwarm')

plt.title("Anomaly Detection in Cows")

plt.show()
```

App development code

Main Activity Java Code

```
package com.example.animalwashuka;

import android.content.Intent;

import android.os.Bundle;

import android.text.TextUtils;

import android.view.View;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.EditText;

import android.widget.Toast;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import com.google.android.gms.tasks.OnCompleteListener;

import com.google.android.gms.tasks.Task;

import com.google.firebase.auth.AuthResult;

import com.google.firebase.auth.FirebaseAuth;

public class MainActivity extends AppCompatActivity {

  private EditText emailEditText;

  private EditText passwordEditText;

  private Button loginButton;

  private CheckBox rememberMeCheckBox;

  private Button createAccount;

  private FirebaseAuth mAuth;

  @Override

  protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

    // Initialize FirebaseAuth

    mAuth = FirebaseAuth.getInstance();

    // Initialize views

    emailEditText = findViewById(R.id.username);

    passwordEditText = findViewById(R.id.password_toggle);

    loginButton = findViewById(R.id.loginButton);

    createAccount = findViewById(R.id.createAccount);

    // Set onClick listener for Login button

loginButton.setOnClickListener(new View.OnClickListener() {

      @Override

      public void onClick(View v) {

        loginUser();

      }

    });

    // Set onClick listener for Create Account button

createAccount.setOnClickListener(new View.OnClickListener() {

      @Override

      public void onClick(View v) {
```

```
                Intent intent = new
Intent(MainActivity.this,
register.class);

                startActivity(intent);

                finish();

            }

        });

    }

    private void loginUser() {

        String email =
emailEditText.getText().toString().trim
();

        String password =
passwordEditText.getText().toString().t
rim();

        if (TextUtils.isEmpty(email)) {


Toast.makeText(MainActivity.this,
"Email is required",
Toast.LENGTH_SHORT).show();

            return;

        }

        if (TextUtils.isEmpty(password))
{


Toast.makeText(MainActivity.this,
"Password is required",
Toast.LENGTH_SHORT).show();

            return;

        }

        // Sign in with email and
password without email verification
check


mAuth.signInWithEmailAndPassword(
email, password)


.addOnCompleteListener(this, new
OnCompleteListener<AuthResult>() {

                    @Override

                    public void
onComplete(@NonNull
Task<AuthResult> task) {

                        if (task.isSuccessful()) {

                            // Sign in successful,
no email verification required


Toast.makeText(MainActivity.this,
"Log in successful.",
Toast.LENGTH_SHORT).show();

                            // Redirect to
DashboardActivity

                            Intent intent = new
Intent(MainActivity.this,
Dashboard.class);

                            startActivity(intent);

                            finish();

                        } else {

                            // If sign in fails,
display a message to the user


Toast.makeText(MainActivity.this,
"Log in failed. Please check your
credentials and try again.",
Toast.LENGTH_SHORT).show();

                        }  } });

    }}
```

MainActivity XML

```
<?xml version="1.0" encoding="utf-
8"?>

<LinearLayout
xmlns:android="http://schemas.android
.com/apk/res/android"


xmlns:app="http://schemas.android.co
m/apk/res-auto"


xmlns:tools="http://schemas.android.c
om/tools"


android:layout_width="match_parent"


android:layout_height="match_parent"


android:background="@drawable/pay
ment"

    android:gravity="center"

    android:orientation="vertical"

    tools:context=".MainActivity">

    <TextView

        android:id="@+id/LogInText"


android:layout_width="match_parent"


android:layout_height="wrap_content"

        android:text="Welcome back!"

        android:textAlignment="center"


android:layout_marginTop="10dp"

        android:textColor="#0329FF"

        android:textSize="30dp"

        android:alpha="0.6"

        android:textStyle="normal" />

    <EditText

        android:id="@+id/username"


android:layout_width="match_parent"

        android:layout_height="55dp"


android:layout_marginTop="40dp"


android:background="@drawable/cust
om_edittext"


android:drawableLeft="@drawable/bas
eline_person_24"

        android:drawablePadding="10dp"

        android:hint="Email"

        android:padding="10dp"


android:textColor="@color/black"


android:textColorHighlight="@color/b
lack"

        android:textSize="20dp" />

    <EditText


android:id="@+id/password_toggle"


android:layout_width="match_parent"

        android:layout_height="65dp"


android:layout_marginTop="40dp"


android:background="@drawable/cust
om_edittext"


android:drawableLeft="@drawable/bas
eline_lock_24"

        android:drawablePadding="20dp"

        android:hint="password"


android:inputType="textPassword"

        android:padding="20dp"


android:textColor="@color/black"


android:textColorHighlight="#5814D1
"

        android:textSize="20dp" />

    <Button

        android:id="@+id/loginButton"
```

```
android:layout_width="match_parent"
        android:layout_height="50dp"

android:layout_marginTop="10dp"

android:backgroundTint="#9C27B0"

android:onClick="launchDashboard"
        android:text="Log In"
        android:textSize="20dp"
        app:cornerRadius="20dp"
        tools:visibility="visible" />
    <CheckBox

android:layout_width="wrap_content"

android:layout_height="wrap_content"
        android:text="Remember me?"

android:layout_centerVertical="true"/>
    <TextView

android:layout_width="wrap_content"

android:layout_height="wrap_content"
        android:text="Or no account?"
        android:textSize="25dp"
        android:alpha="0.8"

android:layout_centerVertical="true"/>
    <LinearLayout

android:layout_width="wrap_content"

android:layout_height="wrap_content"
>
    </LinearLayout>
    <Button

android:layout_width="match_parent"
        android:layout_height="48dp"

android:background="@color/black"
        android:id="@+id/createAccount"
        android:height="50dp"

android:layout_gravity="center|bottom
"
        android:text="Click here to create
account"

android:textColor="@color/white"
        app:cornerRadius="50dp"
        tools:visibility="visible" />
</LinearLayout>
```

Dashboard Java code

```
package com.example.animalwashuka;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import
androidx.appcompat.app.AppCompatA
ctivity;
public class Dashboard extends
AppCompatActivity {
    private Button animalsButton;
    private Button settingsButton;
    private Button logoutButton;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_dash
board);
        // Initialize buttons
        animalsButton =
findViewById(R.id.animalsID);
        settingsButton =
findViewById(R.id.settings);
        logoutButton =
findViewById(R.id.Logout);
        // Set click listeners

animalsButton.setOnClickListener(ne
w View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Open AnimalsActivity
                Intent intent = new
Intent(Dashboard.this, Animals.class);
                startActivity(intent);
            } });

settingsButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Open SettingsActivity
                Intent intent = new
Intent(Dashboard.this, Settings.class);
                startActivity(intent);
            }  });

logoutButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Open MainActivity and
clear activity stack
                Intent intent = new
Intent(Dashboard.this,
MainActivity.class);

intent.addFlags(Intent.FLAG_ACTIVI
TY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_NEW_TAS
K |
Intent.FLAG_ACTIVITY_CLEAR_T
ASK);
                startActivity(intent);
                finish(); // Optional: Finish
this activity
            }  });
    }}
```

Dashbord XML

```
<?xml version="1.0" encoding="utf-
8"?>
<RelativeLayout
xmlns:android="http://schemas.android
.com/apk/res/android"

xmlns:app="http://schemas.android.co
m/apk/res-auto"

xmlns:tools="http://schemas.android.c
om/tools"

android:layout_width="match_parent"

android:layout_height="match_parent"
```

```
android:background="@drawable/hd_geometric_light_lines_pastel_sh_stripes_warm"
    android:layout_gravity="center"
    android:orientation="vertical"
    tools:context=".Dashboard">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:text="My Dashboard"
        android:textSize="30dp"
        android:textStyle="bold"
        android:id="@+id/Dashboardtop"
        android:textColor="#025104"
        android:layout_alignParentTop="true"
        android:layout_marginTop="1dp">
    </TextView>
    <ImageView
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:background="@drawable/baseline_home_24"
        android:layout_alignParentLeft="true"
        android:layout_below="@id/Dashboardtop"
        android:layout_marginTop="50dp">
    </ImageView>
    <Button
        android:id="@+id/animalsID"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="ANIMALS"
        android:textSize="25dp"
        android:textColor="@color/black"
        android:textStyle="italic|bold"
        android:background="@color/teal_200"
        android:layout_below="@id/Dashboardtop"
        android:layout_marginRight="20dp"
        android:layout_marginTop="50dp"
        android:layout_marginLeft="45dp">
    </Button>
    <ImageView
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:background="@drawable/baseline_settings_24"
        android:layout_alignParentLeft="true"
        android:layout_marginTop="30dp"
        android:layout_below="@id/animalsID">
    </ImageView>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/settings"
        android:text="SETTINGS"
        android:background="@color/white"
        android:textColor="@color/black"
        android:textSize="25dp"
        android:textStyle="bold|italic"
        android:layout_below="@id/animalsID"
        android:layout_marginTop="30dp"
        android:layout_marginLeft="45dp"
        android:layout_marginRight="20dp">
    </Button>
    <ImageView
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_below="@id/settings"
        android:layout_marginTop="30dp"
        android:layout_alignParentLeft="true"
        android:background="@drawable/baseline_lock_24"
        >

    </ImageView>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/Logout"
        android:background="@color/teal_200"
        android:text="LOG OUT"
        android:textStyle="italic|bold"
        android:textColor="@color/black"
        android:textSize="25dp"
        android:layout_below="@id/settings"
        android:layout_marginTop="30dp"
        android:layout_marginLeft="45dp"
        android:layout_marginRight="20dp">

    </Button>
</RelativeLayout>
```

Animal Java code

```
package com.example.animalwashuka;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

```
import
androidx.appcompat.app.AppCompatA
ctivity;


public class Animals extends
AppCompatActivity {


    private Button cattleButton,
sheepButton, goatsButton,
chickenButton, addAnimalButton;


    @Override

    protected void onCreate(Bundle
savedInstanceState) {


super.onCreate(savedInstanceState);


setContentView(R.layout.activity_ani
mals);

        // Initialize buttons

        cattleButton =
findViewById(R.id.cattle);

        sheepButton =
findViewById(R.id.sheep);

        goatsButton =
findViewById(R.id.goats);

        chickenButton =
findViewById(R.id.chicken);

        addAnimalButton =
findViewById(R.id.addAnimal);

        // Set onClick listeners for each
button


cattleButton.setOnClickListener(new
View.OnClickListener() {

            @Override

            public void onClick(View v) {

                // Navigate to the Cattle
activity

                Intent intent = new
Intent(Animals.this, Cattle.class);

                startActivity(intent);

            }

        });

        // Implement sheep, goats, and
chicken buttons similarly if needed


sheepButton.setOnClickListener(new
View.OnClickListener() {

            @Override

            public void onClick(View v) {

                // Code to navigate to Sheep
activity

                Intent intent = new
Intent(Animals.this, Sheep.class);

                startActivity(intent);

            }

        });



goatsButton.setOnClickListener(new
View.OnClickListener() {

            @Override

            public void onClick(View v) {

                // Code to navigate to Goats
activity

                Intent intent = new
Intent(Animals.this, Goats.class);

                startActivity(intent);

            }

        });



chickenButton.setOnClickListener(ne
w View.OnClickListener() {

            @Override

            public void onClick(View v) {

                // Code to navigate to
Chicken activity

                Intent intent = new
Intent(Animals.this, Chicken.class);

                startActivity(intent);

            }

        });



addAnimalButton.setOnClickListener(
new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                // Code to navigate to Add
Animal activity

                Intent intent = new
Intent(Animals.this, AddAnimal.class);

                startActivity(intent);

            }  });

    }}
```

Animals XML

```
<?xml version="1.0" encoding="utf-
8"?>

<RelativeLayout
xmlns:android="http://schemas.android
.com/apk/res/android"


xmlns:app="http://schemas.android.co
m/apk/res-auto"


xmlns:tools="http://schemas.android.c
om/tools"


android:layout_width="match_parent"


android:layout_height="match_parent"

    android:orientation="vertical"


android:background="@drawable/pay
ment"

    tools:context=".Animals">

    <TextView


android:layout_width="match_parent"

        android:layout_height="50dp"

        android:text="ANIMALS"

        android:id="@+id/animaltypes"

        android:textStyle="italic|bold"


android:textColor="@color/black"

        android:textSize="30dp"

        >

    </TextView>

    <Button


android:layout_width="match_parent"


android:layout_height="wrap_content"

        android:id="@+id/cattle"

        android:text="Cattle"

        android:textSize="20dp"

        android:textStyle="italic|bold"


android:textColor="@color/white"


android:layout_marginTop="50dp"


android:layout_below="@id/animaltyp
es"
```

```
        >
    </Button>
    <Button
android:layout_width="match_parent"
android:layout_height="wrap_content"
        android:id="@+id/sheep"
android:layout_below="@id/cattle"
android:layout_marginTop="20dp"
        android:text="Sheep"
        android:textSize="20dp"
android:textColor="@color/white"
        android:textStyle="italic|bold"
        >
    </Button>
    <Button
android:layout_width="match_parent"
android:layout_height="wrap_content"
        android:id="@+id/goats"
android:layout_below="@id/sheep"
android:layout_marginTop="20dp"
        android:text="Goats"
android:textColor="@color/white"
        android:textStyle="italic|bold"
        android:textSize="20dp">
    </Button>
    <Button
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_below="@id/goats"
android:layout_marginTop="20dp"
        android:text="Chicken"
android:textColor="@color/white"
        android:textStyle="italic|bold"
        android:textSize="20dp"
        android:id="@+id/chicken">
    </Button>
    <Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/chicken"
        android:text="ADD ANIMAL"
android:textColor="@color/black"
        android:textStyle="normal|bold"
        android:textSize="20dp"
android:layout_marginTop="20dp"
        android:id="@+id/addAnimal"
android:layout_centerHorizontal="true
">

    </Button>

</RelativeLayout>
```

Cattle Java code

```
package com.example.animalwashuka;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import
androidx.appcompat.app.AppCompatA
ctivity;
import
com.amazonaws.auth.CognitoCaching
CredentialsProvider;
import
com.amazonaws.mobile.client.AWSM
obileClient;
import
com.amazonaws.regions.Regions;
import
com.amazonaws.services.dynamodbv2.
AmazonDynamoDBClient;
import
com.amazonaws.services.dynamodbv2.
document.DynamoDB;
import
com.amazonaws.services.dynamodbv2.
document.Item;
import
com.amazonaws.services.dynamodbv2.
document.Table;
import
com.amazonaws.services.dynamodbv2.
document.spec.ScanSpec;
import java.util.Iterator;


public class Cattle extends
AppCompatActivity {
    private LinearLayout
cowsContainer;
    private Button addCattleButton;
    private DynamoDB dynamoDB;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_cattl
e);
        cowsContainer =
findViewById(R.id.cowsContainer);
        addCattleButton =
findViewById(R.id.AddCattle);
        // Initialize AWS Mobile Client

AWSMobileClient.getInstance().initiali
ze(this).execute();
        // Configure Cognito credentials
provider with region

CognitoCachingCredentialsProvider
credentialsProvider = new
CognitoCachingCredentialsProvider(
                getApplicationContext(),

"YOUR_COGNITO_IDENTITY_PO
OL_ID", // Replace with your Cognito
Identity Pool ID
                Regions.US_EAST_1 //
Replace with your DynamoDB region,
e.g., US_EAST_1
        );


        // Create a DynamoDB client with
region configuration
```

```
        AmazonDynamoDBClient
dbClient = new
AmazonDynamoDBClient(credentials
Provider);

dbClient.setRegion(com.amazonaws.re
gions.Region.getRegion(Regions.US_
EAST_1)); // Set region explicitly

        dynamoDB = new
DynamoDB(dbClient);

        // Retrieve and display cows from
DynamoDB

        displayCows();

        // Add Cattle Button functionality

addCattleButton.setOnClickListener(n
ew View.OnClickListener() {

            @Override

            public void onClick(View v) {

                Intent intent = new
Intent(Cattle.this, plus.class);

                startActivity(intent);

            }  });

    }

    private void displayCows() {

        // Reference the DynamoDB table

        Table table =
dynamoDB.getTable("CowData");

        // Scan the table to fetch all cow
records

        new Thread(() -> {

            try {

                ScanSpec scanSpec = new
ScanSpec();

                Iterator<Item> iterator =
table.scan(scanSpec).iterator();

                runOnUiThread(() -> {

cowsContainer.removeAllViews(); //
Clear existing views

                    while (iterator.hasNext())
{

                        Item item =
iterator.next();

                        // Get cow details and
create a Cow object

                        Cow cow = new Cow(

item.getString("CowID"),

item.getString("Name"),

                            item.getInt("Age"),

item.getString("Breed")

                        );

                        // Create a button for
each cow

                        Button cowButton =
new Button(Cattle.this);

cowButton.setText(cow.getName());

cowButton.setTextSize(20);

cowButton.setPadding(10, 10, 10, 10);

cowButton.setOnClickListener(v -> {

                            // Start activity for
detailed cow view (SpecificCow)

                            Intent intent = new
Intent(Cattle.this, SpecificCow.class);

intent.putExtra("cowId",
cow.getCowID());

                            startActivity(intent);

                        });

                        // Add button to the
container

cowsContainer.addView(cowButton);

                    }

                });

            } catch (Exception e) {

                e.printStackTrace();

            }

        }).start();

    }}
```

Cattle XML

```
<?xml version="1.0" encoding="utf-
8"?>

<ScrollView
xmlns:android="http://schemas.android
.com/apk/res/android"

xmlns:app="http://schemas.android.co
m/apk/res-auto"

xmlns:tools="http://schemas.android.c
om/tools"

android:layout_width="match_parent"

android:layout_height="match_parent"

android:background="@drawable/hd_
geometric_light_lines_pastel_sh_stripe
s_warm"

    tools:context=".Cattle">

    <LinearLayout

android:layout_width="match_parent"

android:layout_height="wrap_content"

        android:orientation="vertical"

        android:padding="16dp">

        <TextView

android:id="@+id/cattleHeader"

android:layout_width="match_parent"

android:layout_height="wrap_content"

            android:text="My Cattle"

            android:textSize="25sp"

            android:textStyle="bold"

            android:textColor="#025104"

android:layout_marginTop="10dp" />


        <LinearLayout

android:id="@+id/cowsContainer"

android:layout_width="match_parent"

android:layout_height="wrap_content"

            android:orientation="vertical"

android:layout_marginTop="16dp" />

        <!-- Add Cow Button -->

        <Button

            android:id="@+id/AddCattle"

android:layout_width="match_parent"

android:layout_height="wrap_content"
```

```
android:background="@color/teal_200
"
            android:text="Add Cow"

android:textColor="@color/black"
            android:textSize="20sp"
            android:textStyle="italic|bold"

android:layout_marginTop="16dp" />
    </LinearLayout>
</ScrollView>
```

Specific Cow Java Code

```
package com.example.animalwashuka;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import
androidx.appcompat.app.AppCompatA
ctivity;
import
com.amazonaws.mobile.client.AWSM
obileClient;
import
com.amazonaws.auth.CognitoCaching
CredentialsProvider;
import
com.amazonaws.regions.Regions;
import
com.amazonaws.services.dynamodbv2.
AmazonDynamoDBClient;
//import
com.amazonaws.services.dynamodbv.
R;
import
com.amazonaws.services.dynamoDB2.
document.DynamoDB;
//import
com.amazonaws.services.dynamodbv2.
document.Table;
import
com.amazonaws.services.dynamodbv2.
document.spec.GetItemSpec;
import
com.amazonaws.services.dynamodbv2.
document.utils.ValueMap;
import
com.amazonaws.services.dynamodbv2.
model.GetItemRequest;
import
com.amazonaws.services.dynamodbv2.
model.GetItemResult;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;


public class SpecificCow extends
AppCompatActivity {
    private TextView cowName,
healthIndicator, timestamp,
temperature, activityMagnitude, status;
    private Button deleteCowButton;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_spec
ific_cow);
        // Initialize UI components
        cowName =
findViewById(R.id.cowName);
        healthIndicator =
findViewById(R.id.healthIndicator);
        timestamp =
findViewById(R.id.timestamp);
        temperature =
findViewById(R.id.temperature);
        activityMagnitude =
findViewById(R.id.activityMagnitude)
;
        status =
findViewById(R.id.status);
        deleteCowButton =
findViewById(R.id.deleteCowButton);
        // Fetch data from DynamoDB
(assuming cowId is passed in an Intent)
        String cowId =
getIntent().getStringExtra("COW_ID")
;


        // Fetch details from DynamoDB
        fetchCowDetails(cowId);
        // Set timestamp to current time
from phone
        setTimestamp();
        // Handle delete cow button click
(optional)

deleteCowButton.setOnClickListener(
view -> {
            // Logic to delete the cow from
DynamoDB
            deleteCow(cowId);
        });
    }
    private void fetchCowDetails(String
cowId) {
        // Initialize AWS DynamoDB
client
        DynamoDB dynamoDB = new
DynamoDB(AWSMobileClient.getInst
ance());
        Table table =
dynamoDB.getTable("Cattle");
        // Query the DynamoDB table for
the specific cow using its cowId
        GetItemSpec spec = new
GetItemSpec().withPrimaryKey("cowI
D", cowId);
        try {
            // Fetch the cow data from
DynamoDB

com.amazonaws.services.dynamodbv2.
document.Item item =
table.getItem(spec);
            if (item != null) {
                // Populate UI elements with
the retrieved data

cowName.setText(item.getString("cow
Name"));

healthIndicator.setText(item.getString(
"healthStatus"));

temperature.setText("Temperature: " +
item.getString("temperature"));

activityMagnitude.setText("Activity
Magnitude: " +
item.getString("activityMagnitude"));
                status.setText("Status: " +
item.getString("status"));
            } else {
                // Show a message if the cow
data is not found
                Toast.makeText(this, "Cow
data not found",
Toast.LENGTH_SHORT).show();
            }
        } catch (Exception e) {
            // Handle any errors
            Toast.makeText(this, "Error
fetching data: " + e.getMessage(),
Toast.LENGTH_SHORT).show();

        }

    }

    private void setTimestamp() {

        // Set the timestamp to the current
date and time of the phone

        SimpleDateFormat sdf = new
SimpleDateFormat("hh:mm a, dd
MMM yyyy", Locale.getDefault());

        String currentTime =
sdf.format(new Date());

        timestamp.setText(currentTime);

    }

    private void deleteCow(String
cowId) {

        // Logic to delete the cow from
DynamoDB

        try {

            // Assume that the delete
functionality is implemented

            // For example, using a
DynamoDB delete request

            DynamoDB dynamoDB = new
DynamoDB(AWSMobileClient.getInst
ance());

            Table table =
dynamoDB.getTable("Cattle");

            table.deleteItem("cowID",
cowId);

            Toast.makeText(this, "Cow
deleted successfully",
Toast.LENGTH_SHORT).show();

        } catch (Exception e) {

            // Handle any errors during
deletion

            Toast.makeText(this, "Error
deleting cow: " + e.getMessage(),
Toast.LENGTH_SHORT).show();

        } }}
```

SpecificCow XML

```
<?xml version="1.0" encoding="utf-
8"?>

<ScrollView
xmlns:android="http://schemas.android
.com/apk/res/android"


xmlns:tools="http://schemas.android.c
om/tools"


android:layout_width="match_parent"


android:layout_height="match_parent"


android:background="@drawable/hd_
geometric_light_lines_pastel_sh_stripe
s_warm"

    tools:context=".SpecificCow">

    <LinearLayout


android:layout_width="match_parent"


android:layout_height="wrap_content"

        android:orientation="vertical"

        android:padding="16dp">

        <!-- Cow Name -->

        <TextView

            android:id="@+id/cowName"


android:layout_width="match_parent"


android:layout_height="wrap_content"

            android:text=""

            android:textSize="22sp"

            android:textStyle="bold"

            android:textColor="#000000"


android:layout_marginBottom="8dp"
/>



        <!-- Health Indicator -->

        <TextView


android:id="@+id/healthIndicator"


android:layout_width="match_parent"


android:layout_height="wrap_content"

            android:text=""

            android:textSize="20sp"

            android:textStyle="bold"

            android:textColor="#4CAF50"


android:layout_marginBottom="16dp"
/>

        <!-- Timestamp -->

        <TextView

            android:id="@+id/timestamp"


android:layout_width="match_parent"


android:layout_height="wrap_content"

            android:text=""

            android:textSize="18sp"

            android:textColor="#000000"


android:layout_marginBottom="8dp"
/>

        <!-- Temperature -->

        <TextView

            android:id="@+id/temperature"


android:layout_width="match_parent"


android:layout_height="wrap_content"

            android:text=""

            android:textSize="18sp"

            android:textColor="#000000"


android:layout_marginBottom="8dp"
/>

        <!-- Activity Magnitude -->

        <TextView


android:id="@+id/activityMagnitude"


android:layout_width="match_parent"


android:layout_height="wrap_content"

            android:text=""

            android:textSize="18sp"

            android:textColor="#000000"


android:layout_marginBottom="8dp"
/>



        <!-- Status -->

        <TextView

            android:id="@+id/status"


android:layout_width="match_parent"


android:layout_height="wrap_content"

            android:text=""

            android:textSize="18sp"

            android:textColor="#000000"
```

```
android:layout_marginBottom="16dp"
/>

    <!-- Delete Cow Button -->

    <Button

android:id="@+id/deleteCowButton"

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:background="@color/teal_200
"

        android:text="Delete Cow"

android:textColor="@color/black"

        android:textSize="18sp"

android:layout_marginTop="16dp" />

  </LinearLayout>

</ScrollView>
```

Register Java Code

```
package com.example.animalwashuka;

import
android.annotation.SuppressLint;

import android.content.Intent;

import android.os.Bundle;

import android.text.TextUtils;

import android.util.Patterns;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ProgressBar;

import android.widget.Toast;

import androidx.annotation.NonNull;

import
androidx.appcompat.app.AppCompatA
ctivity;

import
com.google.android.gms.tasks.OnCom
pleteListener;

import
com.google.android.gms.tasks.Task;

import
com.google.firebase.auth.AuthResult;

import
com.google.firebase.auth.FirebaseAuth
;

import
com.google.firebase.auth.FirebaseUser
;

public class register extends
AppCompatActivity {

  private EditText fullNameEditText,
phoneNumberEditText, emailEditText,
passwordEditText;

  private Button registerButton,
signInButton;

  private FirebaseAuth mAuth;

  private ProgressBar progressBar;

  @SuppressLint("MissingInflatedId")

  @Override

  protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_regis
ter);

    // Initialize Firebase Auth

    mAuth =
FirebaseAuth.getInstance();

    // Initialize views

    fullNameEditText =
findViewById(R.id.fullname);

    phoneNumberEditText =
findViewById(R.id.phonenumber);

    emailEditText =
findViewById(R.id.emailRegistration);

    passwordEditText =
findViewById(R.id.password_toggle);

    registerButton =
findViewById(R.id.registerBtn);

    signInButton =
findViewById(R.id.signInButton);

    progressBar =
findViewById(R.id.progressBar);

    // Register button click listener

registerButton.setOnClickListener(new
View.OnClickListener() {

      @Override

      public void onClick(View
view) {

        registerUser();

      }  });

    // Sign in button click listener

signInButton.setOnClickListener(new
View.OnClickListener() {

      @Override

      public void onClick(View
view) {

        // Redirect to Main Activity
(Login)

        startActivity(new
Intent(register.this,
MainActivity.class));

      }  });

  }

  private void registerUser() {

    String fullName =
fullNameEditText.getText().toString().t
rim();

    String phoneNumber =
phoneNumberEditText.getText().toStri
ng().trim();

    String email =
emailEditText.getText().toString().trim
();

    String password =
passwordEditText.getText().toString().t
rim();

    // Validation

    if (TextUtils.isEmpty(fullName) ||
!fullName.matches("[a-zA-Z\\s]+")) {

fullNameEditText.setError("Please
enter a valid full name");

fullNameEditText.requestFocus();

      return;

    }

    if
(TextUtils.isEmpty(phoneNumber) ||
phoneNumber.length() != 10) {

phoneNumberEditText.setError("Pleas
e enter a valid 10-digit phone
number");

phoneNumberEditText.requestFocus();

      return;

    }

    if (TextUtils.isEmpty(email) ||
!Patterns.EMAIL_ADDRESS.matcher(
email).matches()) {

      emailEditText.setError("Please
enter a valid email");
```

```
            emailEditText.requestFocus();

            return;

        }

        if (TextUtils.isEmpty(password) ||
password.length() < 6) {

passwordEditText.setError("Password
must be at least 6 characters");

passwordEditText.requestFocus();

            return;

        }

        // Show progress bar

progressBar.setVisibility(View.VISIBL
E);

        // Check if the email is already in
use

mAuth.fetchSignInMethodsForEmail(e
mail).addOnCompleteListener(new
OnCompleteListener<com.google.fireb
ase.auth.SignInMethodQueryResult>()
{

            @Override

            public void
onComplete(@NonNull
Task<com.google.firebase.auth.SignIn
MethodQueryResult> task) {

                if (task.isSuccessful()) {

                    boolean emailExists =
!task.getResult().getSignInMethods().i
sEmpty();

                    if (emailExists) {

                        // Email already in use

progressBar.setVisibility(View.GONE)
;

emailEditText.setError("This email is
already registered");

emailEditText.requestFocus();

                    } else {

                        // Proceed with
registration

registerWithEmailAndPassword(email,
password);

                    }

                } else {

                    // Handle error

progressBar.setVisibility(View.GONE)
;

Toast.makeText(register.this, "Error
checking email: " +
task.getException().getMessage(),
Toast.LENGTH_SHORT).show();

                } } });

    }

    private void
registerWithEmailAndPassword(String
email, String password) {

mAuth.createUserWithEmailAndPass
word(email, password)

.addOnCompleteListener(register.this,
new
OnCompleteListener<AuthResult>() {

                    @Override

                    public void
onComplete(@NonNull
Task<AuthResult> task) {

progressBar.setVisibility(View.GONE)
;

                        if (task.isSuccessful()) {

                            // Send verification
email

                            FirebaseUser user =
mAuth.getCurrentUser();

                            if (user != null) {

user.sendEmailVerification()

.addOnCompleteListener(new
OnCompleteListener<Void>() {

                                        @Override

                                        public void
onComplete(@NonNull Task<Void>
task) {

                                            if
(task.isSuccessful()) {

Toast.makeText(register.this,
"Registration successful. Please check
your email to verify your account.",
Toast.LENGTH_LONG).show();

                                                //
Redirect to Dashboard Activity

                                                Intent
intent = new Intent(register.this,
Dashboard.class);

startActivity(intent);

finish();

                                            } else {

Toast.makeText(register.this, "Failed to
send verification email: " +
task.getException().getMessage(),
Toast.LENGTH_SHORT).show();

                                            }  }

                                    });

                            }

                        } else {

                            // Registration failed

Toast.makeText(register.this,
"Registration failed: " +
task.getException().getMessage(),
Toast.LENGTH_SHORT).show();

                        } }

                });

    }

}
```

Register XML

```
<ScrollView
xmlns:android="http://schemas.android
.com/apk/res/android"

xmlns:app="http://schemas.android.co
m/apk/res-auto"

xmlns:tools="http://schemas.android.c
om/tools"

android:layout_width="match_parent"

android:layout_height="match_parent"

android:background="@drawable/pay
ment"

    tools:context=".register">

    <LinearLayout

android:layout_width="match_parent"

android:layout_height="wrap_content"

        android:orientation="vertical"

        android:padding="20dp">

        <!-- ProgressBar for registration --
>

        <ProgressBar
```

```
android:id="@+id/progressBar"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:layout_gravity="center"
            android:visibility="gone"
            android:indeterminate="true"
/>
        <TextView

android:layout_width="match_parent"

android:layout_height="wrap_content"
            android:text="Register"

android:textAlignment="center"
            android:textSize="32sp"

android:textColor="@color/black" />
        <!-- Full Name Input field -->
        <EditText
            android:id="@+id/fullname"

android:layout_width="match_parent"
            android:layout_height="48dp"

android:layout_marginTop="20dp"

android:background="@drawable/cust
om_edittext"

android:drawableLeft="@drawable/bas
eline_person_24"

android:drawablePadding="10dp"
            android:hint="Full name"
            android:padding="10dp"

android:textColor="@color/black"

android:textColorHighlight="@color/b
lack"
            android:textSize="20sp" />
        <!-- Phone Number Input Field --
>
        <EditText

android:id="@+id/phonenumber"

android:layout_width="match_parent"
            android:layout_height="48dp"

android:layout_marginTop="20dp"

android:background="@drawable/cust
om_edittext"

android:drawableLeft="@drawable/bas
eline_contact_phone_24"

android:drawablePadding="10dp"
            android:hint="Phone Number"
            android:padding="10dp"

android:textColor="@color/black"

android:textColorHighlight="@color/b
lack"
            android:textSize="20sp" />
        <!-- Email Input Field -->
        <EditText

android:id="@+id/emailRegistration"

android:layout_width="match_parent"
            android:layout_height="48dp"

android:layout_marginTop="20dp"

android:background="@drawable/cust
om_edittext"

android:drawableLeft="@drawable/bas
eline_alternate_email_24"

android:drawablePadding="10dp"
            android:hint="Email"
            android:padding="10dp"

android:textColor="@color/black"

android:textColorHighlight="@color/b
lack"
            android:textSize="20sp" />
        <!-- Password Input Field -->
        <EditText

android:id="@+id/password_toggle"

android:layout_width="match_parent"
            android:layout_height="48dp"

android:layout_marginTop="20dp"

android:background="@drawable/cust
om_edittext"

android:drawableLeft="@drawable/bas
eline_lock_24"

android:drawablePadding="20dp"
            android:hint="Password"

android:inputType="textPassword"
            android:padding="20dp"

android:textColor="@color/black"

android:textColorHighlight="#5814D1
"
            android:textSize="20sp" />
        <!-- Register Button -->
        <Button
            android:id="@+id/registerBtn"

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:layout_marginTop="20dp"

android:background="@color/purple_7
00"
            android:padding="10dp"
            android:text="Register"

android:textColor="@color/white"
            android:textSize="20sp"
            app:backgroundTint="@null"
/>
        <!-- Sign In Button -->
        <Button

android:id="@+id/signInButton"

android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"

android:layout_marginTop="20dp"
            android:text="Sign In"

android:textColor="@color/black"
            android:textSize="18sp" />
    </LinearLayout>
</ScrollView>
```

Contact support Java Code

```
package com.example.animalwashuka;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.widget.Button;
import android.widget.Toast;


import
androidx.appcompat.app.AppCompatA
ctivity;
public class ContactSupport extends
AppCompatActivity {
    private static final String EMAIL =
"riniwachuka2002@gmail.com";
    private static final String
PHONE_NUMBER =
"+254759787739";
    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_cont
act_support);
        Button emailSupportButton =
findViewById(R.id.emailSupport);
        Button callSupportButton =
findViewById(R.id.callSupport);
        // Email support button click
listener

emailSupportButton.setOnClickListene
r(v -> sendEmail());
        // Call support button click
listener

callSupportButton.setOnClickListener(
v -> makePhoneCall());
    }
    private void sendEmail() {
        Intent emailIntent = new
Intent(Intent.ACTION_SENDTO);

emailIntent.setData(Uri.parse("mailto:"
+ EMAIL)); // Only email apps should
handle this

emailIntent.putExtra(Intent.EXTRA_S
UBJECT, "Support Request");

emailIntent.putExtra(Intent.EXTRA_T
EXT, "Please describe your issue:");
        // Check if there's an app that can
handle this intent
        if
(emailIntent.resolveActivity(getPackag
eManager()) != null) {
            startActivity(emailIntent);
        } else {
            Toast.makeText(this, "No email
app found",
Toast.LENGTH_SHORT).show();
        }  }
    private void makePhoneCall() {
        Intent callIntent = new
Intent(Intent.ACTION_DIAL);
        callIntent.setData(Uri.parse("tel:"
+ PHONE_NUMBER)); // The tel URI
scheme
        startActivity(callIntent);
    }}
```

Contact Support XML

```
<ScrollView
xmlns:android="http://schemas.android
.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="match_parent"
    android:padding="16dp"

android:background="@drawable/hd_
geometric_light_lines_pastel_sh_stripe
s_warm">
    <LinearLayout

android:layout_width="match_parent"

android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView

android:layout_width="wrap_content"

android:layout_height="wrap_content"
            android:text="Contact Support"
            android:textSize="24sp"
            android:textStyle="bold"

android:paddingBottom="16dp"/>
        <TextView

android:layout_width="wrap_content"

android:layout_height="wrap_content"
            android:text="If you need
assistance, you can reach out to us via
email or call:"
            android:textSize="16sp"
            android:alpha="0.7"

android:paddingBottom="16dp"/>
        <!-- Email Option -->
        <Button

android:id="@+id/emailSupport"

android:layout_width="match_parent"
            android:layout_height="50dp"
            android:text="Email Support"

android:backgroundTint="@color/mat
erial_dynamic_neutral30"

android:textColor="@color/black"

android:layout_marginLeft="10dp"

android:layout_marginRight="10dp"

android:background="@color/purple_7
00"
            android:textSize="13dp"
            android:padding="12dp"/>
        <!-- Phone Call Option -->
        <Button
            android:id="@+id/callSupport"

android:layout_width="match_parent"
```

```
            android:layout_height="50dp"

android:layout_marginLeft="10dp"

android:layout_marginRight="10dp"

            android:text="Call Support"

android:backgroundTint="@color/mat
erial_dynamic_neutral30"

android:background="@color/purple_7
00"
            android:textSize="13dp"

android:textColor="@color/black"

            android:padding="12dp"

android:layout_marginTop="16dp"/>

    </LinearLayout>

</ScrollView>
```

Settings Java Code

```
package com.example.animalwashuka;

import android.content.Intent;

import android.os.Bundle;

import android.util.Log;

import android.widget.TextView;

import
androidx.appcompat.app.AppCompatA
ctivity;

public class Settings extends
AppCompatActivity {

    private static final String TAG =
"SettingsActivity";

    @Override

    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_setti
ngs);

        try {

            // User Account Settings

            TextView changeEmail =
findViewById(R.id.changeEmail);

changeEmail.setOnClickListener(v ->
{

            startActivity(new
Intent(Settings.this,
ChangeEmail.class));

            });

            TextView changePassword =
findViewById(R.id.changePassword);

changePassword.setOnClickListener(v
-> {

            startActivity(new
Intent(Settings.this,
ChangePassword.class));

            });

            TextView updateProfile =
findViewById(R.id.updateProfile);

updateProfile.setOnClickListener(v ->
{

            startActivity(new
Intent(Settings.this,
UpdateProfile.class));

            });

            // Notifications Settings

            TextView enableNotifications =
findViewById(R.id.enableNotifications
);

enableNotifications.setOnClickListene
r(v -> {

            startActivity(new
Intent(Settings.this,
EnableNotifications.class));

            });

            // Data Settings

            TextView dataSyncFrequency =
findViewById(R.id.dataSyncFrequenc
y);

dataSyncFrequency.setOnClickListener
(v -> {

            showDataSyncOptions();

            });

            // Support & Feedback

            TextView contactSupport =
findViewById(R.id.contactSupport);

contactSupport.setOnClickListener(v -
> {

            startActivity(new
Intent(Settings.this,
ContactSupport.class));

            });

        } catch (Exception e) {

            Log.e(TAG, "Error initializing
settings activity", e);

        }  }

    private void
showDataSyncOptions() {

        String[] options = {"Real-time",
"Every 1 second"};

        new
android.app.AlertDialog.Builder(this)

                .setTitle("Select Data Sync
Frequency")

                .setItems(options, (dialog,
which) -> {

                    if (which == 0) {

setDataSyncFrequency(0); // Real-time

                    } else if (which == 1) {

setDataSyncFrequency(1000); // Every
1 second

                    } })

                .show();

    }

    private void
setDataSyncFrequency(int intervalMs)
{

android.content.SharedPreferences
preferences =
getSharedPreferences("settings",
MODE_PRIVATE);

preferences.edit().putInt("dataSyncFre
quency", intervalMs).apply();

        // Apply the sync frequency to
your data sync service

    }

}
```

Settings XML

```
<ScrollView
xmlns:android="http://schemas.android
.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="match_parent"

    android:padding="16dp"
```

```
android:background="@drawable/hd_geometric_light_lines_pastel_sh_stripes_warm">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <!-- User Account Settings -->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="48dp"
            android:text="User Account Settings"
            android:textSize="18sp"
            android:textStyle="bold"
            android:paddingTop="16dp"
            android:paddingBottom="8dp"/>
        <View
            android:layout_width="match_parent"
            android:layout_height="1dp" />
        <TextView
            android:id="@+id/changeEmail"
            android:layout_width="match_parent"
            android:layout_height="48dp"
            android:text="Change Email"
            android:paddingVertical="12dp"
            android:clickable="true"
            android:focusable="true"
            android:background="?attr/selectableItemBackground"/>
        <TextView
            android:id="@+id/changePassword"
            android:layout_width="match_parent"
            android:layout_height="48dp"
            android:text="Change Password"
            android:paddingVertical="12dp"
            android:clickable="true"
            android:focusable="true"
            android:background="?attr/selectableItemBackground"/>
        <TextView
            android:id="@+id/updateProfile"
            android:layout_width="match_parent"
            android:layout_height="48dp"
            android:text="Update Profile"
            android:paddingVertical="12dp"
            android:clickable="true"
            android:focusable="true"
            android:background="?attr/selectableItemBackground"/>
        <!-- Notifications Settings -->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Notifications Settings"
            android:textSize="18sp"
            android:textStyle="bold"
            android:paddingTop="16dp"
            android:paddingBottom="8dp"/>
        <View
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:background="@color/material_dynamic_neutral30"/>
        <TextView
            android:id="@+id/enableNotifications"
            android:layout_width="match_parent"
            android:layout_height="48dp"
            android:text="Enable/Disable Notifications"
            android:paddingVertical="12dp"
            android:clickable="true"
            android:focusable="true"
            android:background="?attr/selectableItemBackground"/>
        <!-- Data Settings -->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Data Settings"
            android:textSize="18sp"
            android:textStyle="bold"
            android:paddingTop="16dp"
            android:paddingBottom="8dp"/>
        <View
            android:layout_width="match_parent"
            android:layout_height="1dp"
          />
        <TextView
            android:id="@+id/dataSyncFrequency"
            android:layout_width="match_parent"
            android:layout_height="48dp"
            android:text="Data Sync Frequency"
            android:paddingVertical="12dp"
            android:clickable="true"
            android:focusable="true"
            android:background="?attr/selectableItemBackground"/>
        <!-- Support & Feedback -->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="48dp"
            android:text="Support and
Feedback"

            android:textSize="18sp"

            android:textStyle="bold"

            android:paddingTop="16dp"


android:paddingBottom="8dp"/>

        <View


android:layout_width="match_parent"

            android:layout_height="1dp"


android:background="@color/material
_dynamic_neutral30"/>

        <TextView


android:id="@+id/contactSupport"


android:layout_width="match_parent"

            android:layout_height="48dp"

            android:text="Contact Support"


android:paddingVertical="12dp"

            android:clickable="true"

            android:focusable="true"


android:background="?attr/selectableIt
emBackground"/>

    </LinearLayout>

</ScrollView>
```