

Smart farming using IOT devices and AWS cloud services

By

Aartiben Sunilkumar Prajapati

“IV” semester "Master of Computer Application "

Registration No. MCA22425



Project Report submitted to the University
of Mysore in partial fulfilment of the
requirements of “IV” Semester “**Master of Computer Application**” degree
examinations **2024**

University of Mysore,
Manasagangotri, Mysore– 570006

Smart farming using IOT devices and AWS cloud services

By

Aartiben Sunilkumar Prajapati

“IV” semester "Master of Computer Application "

Registration No. MCA22425



Project Report submitted to the University
of Mysore in partial fulfilment of the
requirements of “IV” Semester “**Master of Computer Application**” degree
examinations **2024**

University of Mysore,
Manasagangotri, Mysore– 570006

DECLARATION

I, **Aartiben Sunilkumar Prajapati**, hereby declare that the project report, entitled “**Smart farming using IOT devices and AWS cloud services**”, submitted to the University of mysore in partial fulfilment of the requirements for the award of the Degree of Master of Computer application is submitted to the Directorate of Outreach and Online Programmes, University of Mysore and it has not formed the basis for the award of any Degree/Fellowship or other similar title to any candidate of any university.

Place:

Date:

Signature of the Student:

ACKNOWLEDGEMENT

I am writing to express my sincere appreciation for the opportunity to complete my project, **"Smart farming using IOT devices and AWS cloud services"** through the University of Mysore. Undertaking this project as part of my university coursework has been invaluable in enhancing my knowledge and skills in the field of IT. The curriculum provided by the university has equipped me with a solid foundation, enabling me to conceptualize and execute the project effectively. Furthermore, I am grateful for the practical experience I have gained from working in the IT industry. This experience has complimented my academic learning, allowing me to apply theoretical concepts to real-world scenarios with confidence.

1.1 Introduction	9
1.1.1 Smart Farming.....	9
1.1.2 Traditional Farming.....	9
1.1.3 Key differences:	10
1.1.4 Effect of Temperature on Crop	10
1.1.5 Effects of humidity on Crops.....	11
1.1.6 Effects of NO ₂ on Crops	13
1.1.7 Effects of fertilizers on Crops.....	13
1.1.8 PH level of Soil.....	14
1.1.9 Irrigation in Agriculture	15
1.1.10 Overuse of water and fertilizers	16
1.1.11 Need for smart farming arises from various factors, including:	17
1.2 Problem Statement.....	18
1.2.1 The system should:	18
1.2.2 Goals:.....	18
1.2.3 Constraints:	18
1.3 Background and context.....	19
1.3.1 Importance of Smart Farming	19
1.4 Objectives.....	22
1.4.1 Use IOT Sensors for precision Agriculture	22
1.4.1.1 Soil Moisture Monitoring.....	22
1.4.1.2 Temperature and Humidity Monitoring	22
1.4.1.3 Weather Monitoring:	22
1.4.1.4 Equipment Monitoring:	22
1.4.2 Use IOT devices to autonomous and remote control.....	22
1.4.3 Provide predictive and statical analysis from data	22
1.4.3.1 Soil Moisture Analysis:	22
1.4.3.2 weather data Analysis:	23
1.5 Methodology	23
1.5.1 What is the IoT?	24
1.5.2 Examples of IoT devices:	24
1.6 Smart Farming Infrastructure.....	25

1.6.1 Hardware Components	25
1.6.1.1 Sensors.....	25
1.6.1.2 Microcontrollers:	25
1.6.1.3 Communication Modules	26
1.6.1.4. Actuators	26
1.6.1.5 Power Supply	26
1.6.2 Software Components:	26
1.6.2.1 Operating System	26
1.6.2.2 Programming Language.....	26
1.6.2.3 IoT Platform:.....	26
1.6.2.4 Data Analytics	26
1.6.3 System Design:.....	26
1.6.3.1 Sensor Node	26
1.6.4 Smart Farming Applications:.....	27
2.1 Literature Review	27
2.1.1 Early Studies	28
2.1.2 Recent Advances.....	28
2.1.4 Case Studies	28
2.1.5 Challenges and Future Directions	29
2.2 Proposed Method	30
2.2.1 System Design:.....	30
2.2.2 System Components	31
2.2.2.1 Hardware	31
2.2.2.2 Software	31
2.2.2.3 Proof of concept design.....	31
.....	31
2.3 Hardware	34
2.3.1 ESP32 Microcontroller.....	34
2.3.2 Sensors.....	35
2.3.2.1 Soil Moisture Sensor	35
2.3.2.2 ESP32 code for Moisture sensor.....	39
2.3.2.3 Temperature and Humidity Sensor	40

3.2.2.4 ESP32 code for DHT11 Sensor Interfacing.....	46
2.3.2.5 Light Intensity Sensor	47
3.2.3.6 ESP32 interface code for light sensor	52
2.3.2.7 Rain Fall Sensor	53
3.2.3.8 ESP32 Code for Rainfall sensor	58
2.3.3 ESP32 Firmware.....	59
2.3.3.1 Firmware Development:	60
2.3.3.2 Firmware Updates	60
2.3.4.1 Data collection	60
2.3.4.2 The scheduler.....	63
2.3.4.3 Wifi connection and AWS connection	64
2.3.4.4 ESP32 IoT Core Connectivity.....	66
2.3.4.5 MQTT Connection Details:	66
2.3.4.6 Webserver for remote control and OTA update	68
2.3.4.7 ESP32 WebServer Code.....	69
2.3.4.8 Login Page for ESP-32 webserver	70
2.3.4.9 View sensor data in real-time from ESP32 webserver	72
2.3.4.10 Control and Monitor Equipements in real-time from ESP32 webserver	75
2.4 AWS cloud services	78
2.4.1 Steps to create an IoT Core thing in AWS.....	79
2.4.2 Testing AWS IOT core device connection	81
Subscribing Sensor Data to AWS Dshboard	81
2.4.3.1 AWS Lambda.....	83
2.4.3.2 AWS Lambada Trigger with AWS API gateway Setups	85
2.4.4.2 AWS Lambada Code	86
2.4.4 Store / retrieve records in DynamoDB	88
2.4.1.1 Sensor Data from DynamoDB tables	90
3.1.1 statistical analysis in smart farming using IoT.....	90
3.1.2 Statistical Analysis Techniques	91
3.2. Plotly	92
3.2.1 Architecture of Data analysis module	92

3.2.2 Code to store Table data to local Device	93
3.2.3 Plotting Charts from Data tables	94
3.2.3.1 Types of Charts supported by Plotly	95
3.2.4 Creating a scatter plot from IoT sensor data using Plotly	96
3.2.5 Dashboard from Data collected from IOT sensor	97
3.2.5.1 Dashboard	98
3.2.5 Dashboard Filters	99
3.3 Data Analytics	100
5.1 Conclusion	102
5.1.1 Remove Human error	102
5.1.2 Data collection and analysis	103
5.1.3 More control over process	103
5.1.4 Enhanced product quality and yield	103
5.1.5 Cost management	103
5.1.6 Improved efficiency	103
5.1.7 Reduced human resources	103
5.1.8 Early disease detection and prevention	103
5.1.9 Security	103
Bibliography	105

Chapter 1

1.1 Introduction

Smart farming, also known as precision agriculture, is an innovative approach to farming that uses advanced technology to optimize crop yields, reduce waste, and promote sustainability. It involves the use of sensors, drones, satellite imaging, and other digital tools to collect and analyze data on soil conditions, weather patterns, crop health, and other factors that affect agricultural productivity.

Comparison between smart farming and traditional farming

1.1.1 Smart Farming

Technology-driven: Uses advanced technologies like IoT, AI, drones, and sensors to monitor and control farm processes.

Data-driven decision-making: Analyzes data to make informed decisions on irrigation, fertilization, pest control, and harvesting.

Precision agriculture: Optimizes resource usage, reduces waste, and promotes sustainability.

Increased efficiency: Automates tasks, reduces labor, and improves productivity.

Real-time monitoring: Enables remote monitoring of farm conditions, enabling prompt action.

Scalability: Can be applied to small or large farms, with benefits increasing with scale.

Sustainability: Promotes environmentally friendly practices, reducing chemical usage and conserving resources.

1.1.2 Traditional Farming

Labor-intensive: Relies heavily on manual labor for planting, irrigation, and harvesting.

Experience-based decision-making: Decisions based on farmer experience, intuition, and traditional practices.

Broad-acre approach: Applies uniform practices to entire fields, without precise control.

Limited efficiency: Often results in wasted resources, reduced productivity, and lower yields.

Local monitoring: Farmers physically inspect fields to identify issues, potentially delaying action.

Limited scalability: Can be challenging to expand or adapt to changing conditions.

Environmental impact: May involve excessive chemical usage, water waste, and soil degradation.

1.1.3 Key differences:

Technology adoption: Smart farming embraces advanced technologies, while traditional farming relies on established methods.

Decision-making approach: Smart farming uses data analysis, while traditional farming relies on experience and intuition.

Resource optimization: Smart farming prioritizes precision and efficiency, reducing waste and promoting sustainability.

By adopting smart farming practices, farmers can increase efficiency, reduce costs, and promote environmentally friendly agriculture, while traditional farming methods may benefit from incorporating some smart farming elements to improve productivity and sustainability.

1.1.4 Effect of Temperature on Crop

The effect of high-temperature situations leads to a significant reduction in yield. The elevated temperature on crops is expected to have a widespread negative effect as a consequence of global warming. Meanwhile, the global population is rapidly increasing and is predicted to be 11 billion in 2100. An increase in 70% of global food production is a challenging task to feed the increasing population. Increasing the food crop yield is crucial to meet the global food demand and ensuring food security. An increase in high temperature every year due to global warming and an increase in greenhouse gases leads to a rise in temperature. The rise in temperature significantly affects the yield; so, it is important to understand the mechanism and how to counteract high temperature on food crops. It is also important to neutralize the effect of high temperature on food crops and to increase the yield by minimizing the effect of high temperature and developing heat resistant or tolerant variety. It is essential to develop heat-tolerant crops or transgenic food crops that can assure great yield and food security for future generations. It is essential to examine the metabolic, physiological, and molecular mechanisms of food crops to have an enhanced understanding of high temperature and their effects on crops.

Some Studies describes the effects of higher day and night temperatures on crop growth and yield. Temperature effects at different levels of organization - biochemical, physiological, morphological, agronomic and systems - are considered. This is followed by identification of options for germplasm improvement and crop management that may mitigate the adverse effects of higher day and night temperatures.

The maximum threshold temperature for various crops differs. However, high temperatures above 35°C can cause damage to rice crops. Evident injuries

were observed due to high temperatures in different developmental stages. Recent studies exposed that sorghum pistils and pearl millet both are similarly sensitive to high temperatures

1.1.5 Effects of humidity on Crops

Relative humidity:

It is the ratio of actual water vapour content to the saturated water vapour content at a given temperature and pressure expressed in percentage (%).

Diurnal variation in relative humidity:

- Mean maximum relative humidity occurs in the early morning.
- Mean minimum, relative humidity occurs in the early afternoon.
- Low RH in the afternoon is due to expansion of air and thus increases the total water vapour capacity

Distribution of RH

- Maximum RH is in the equatorial region due to high evaporation.
- Decreases towards poles upto 30° N and S due to subsiding air mass.
- RH increases in poles due to low temperature.

Relative humidity (RH) directly influences the water relations of plant and indirectly affects leaf growth, photosynthesis, pollination, occurrence of diseases and finally economic yield.

The dryness of the atmosphere as represented by saturation deficit (100-RH) reduces dry matter production through stomatal control and leaf water potential.

Leaf Growth

- Leaf growth not only depends on synthetic activities resulting from biochemical process but also upon the physical process of cell enlargement.
- Cell enlargement occurs as a result of turgor pressure developed within the cells.
- Turgor pressure is high under RH due to less transpiration. Thus leaf enlargement is high in humid areas

Photosynthesis

- Photosynthesis is indirectly affected by RH. When RH is low, transpiration increases causing water deficits in the plant.
- Water deficits cause partial or full closure of stomata and increase mesophyll resistance blocking entry of carbon dioxide.

Pollination

- Moderately low air humidity is favourable for seed set in many crops, provided soil moisture supply is adequate.
- For example, seed set in wheat was high at 60 per cent RH compared to 80 per cent when water availability in the soil was not limiting.
- At high RH pollen may not be dispersed from the anthers

Pests

- The incidence of insect pests and diseases is high under high humidity conditions.
- High RH favours easy germination of fungal spores on plant leaves.

For example
The blight diseases of potato and tea spread more rapidly under humid conditions. Several insects such as aphids and jassids thrive better under moist conditions.

Grain Yield

Very high or very low RH is not conducive for high grain yield. Under high humidity, RH is negatively correlated with grain yield of maize. The yield reduction was 144 kg/ha with an increase in one per cent of mean monthly RH. Similarly, wheat grain yield is reduced in high RH. It can be attributed to adverse effect of RH on pollination and high incidence of pests. On the contrary, increase in RH during panicle initiation to maturity increased grain yield of sorghum under low humidity conditions due to favourable influence of RH on water relations of plants and photosynthesis. With similar amount of solar radiation, crops that are grown with irrigation gives less yield compared to those grown with equal amount of 'water as rainfall. This is because the dry atmosphere, which is little affected by irrigation, independently suppresses the growth of crops.

Very High Relative humidity:

- Reduces evapotranspiration
- Increases heat load of plants
- Stomatal closure
- Reduced CO₂ uptake
- Reduced transpiration influences translocation of food materials and nutrients.
- Moderately high RH of 60-70% is beneficial.
- Low RH increases the evapotranspiration

1.1.6 Effects of NO₂ on Crops

Two case studies are described in which the possibility of injury to vegetation by NO_x was investigated. In the first study, the very high levels (up to 3500 ppb) of NO_x (mainly NO) in glasshouses have been shown to be inhibitory to plant growth. The effects were not, however, as great as would be expected from such high concentrations. Several reasons why glasshouse crops escape the full impact of the NO_x around them are given. It is noted that the NO_x in the atmosphere of some glasshouses could theoretically supply all the nitrogen required by the crop.

In the second study, effects of NO₂ at concentrations almost two orders of magnitude lower than the NO_x in glasshouses are shown to be important when SO₂ is in the atmosphere at the same time. NO₂ and SO₂ normally accompany one another in urban air, and in rural areas of industrialised countries. The effects of exposure to NO₂ + SO₂ are sometimes greater than would be predicted from the action of the two gases separately. SO₂ has been shown to interfere with the induction of the enzyme nitrite reductase, which reduces NO₂⁻ to NH₃ and may be the main means by which the toxicity of NO₂⁻ is controlled in plant cells. The results of some new dose-response studies of NO₂ + SO₂ mixtures on seedling grasses are presented.

1.1.7 Effects of fertilizers on Crops

due to the inappropriate use of mineral fertilizers (i.e., when used in both excess or deficiency), mostly concerning nitrogenous and phosphate, many productive soils have been thwarted in their ability to function, as shown not only by chemical indicators but also by physical and biological ones. Thus, improper fertilizing technology might have a negative effect on soil health and soil-related ecosystem services. Imbalanced use of chemical fertilizers can alter soil pH, and increase pests attack, acidification, and soil crust, which results in a decrease in soil organic carbon and useful organisms, stunting plant growth and yield, and even leading to the emission of greenhouse gases. Soil health is defined as the capacity of soil to function as a vital living system, within ecosystem and land-use boundaries, to sustain plant and animal health and productivity, and maintain or improve water and air quality. A major challenge for agricultural sustainability is to conserve ecosystem service delivery while optimizing agricultural yields. This Special Issue addresses the task to find a balance between increasing yields using conventional and novel fertilizers, and the maintenance of soil and environmental health as a basis for the sustainable intensification of the agricultural sector

Agriculture is one of the most important components of our society. Farmers and ranchers produce the food and fiber we use every day. Soil is a critical part of successful agriculture and is the original source of the nutrients that we use to grow crops. The nutrients move from the soil into plants that we eat like tomatoes.

Nutrients are also a part of the food animals (like cows) eat. In the end, we benefit from healthy soil. The healthiest soils produce the healthiest and most abundant food supplies. Since the healthiest soils produce the most food, they have also been at the centre of the best communities in history. Ancient Egyptians had fresh nutrients delivered to their fields every year by the flooding of the Nile River. This allowed farmers to use the same soils for a very long time. Some other ancient civilizations had soil that was nutrient poor or didn't hold much water. The health of these soils declined over time, and people had to move away to farm new ground. Soils in dry climates are more likely to see a decline in nutrients, especially if irrigated. The small amounts of salts and other chemicals found in water can build up in the soil over time if not carefully managed. This process is called salinization. Much like freshwater fish are not able to live in the saline ocean, many land plants, including most crops, are not able to grow in saline soil. The most successful civilizations in history have lived on healthy soils and taken good care of the soil so that food production was sustainable. Adding fertilizer is one important way to keep agricultural production systems sustainable. In nature, plants use soil nutrients, and then they die and are decomposed by microorganisms. This returns the nutrients to the soil. In an agricultural setting, the crops take up nutrients, but then are removed from the field so people and livestock can eat them and in turn get the nutrients. This removes nutrients from the field. In order to maintain nutrient levels in soil, it is important to apply fertilizer, whether from natural sources, such as manure, or human-made sources, such as ammonium.

1.1.8 PH level of Soil

The pH of a soil refers to how acid or alkaline the soil is. The letters "pH" means "potential hydrogen." The availability of nutrients is directly affected by soil pH. If the soil's pH is too high or too low, some nutrients become insoluble, limiting the availability of these nutrients to the plant root system.

The acidity-alkalinity scale ranges from 0 to 14. Soils are referred to as being acid, neutral, or alkaline, depending on their pH levels. A pH of 7 is neutral, while a pH lower than 7 is acid, and a pH higher than 7 is alkaline (basic).

A logarithmic scale is used to measure a soil's pH. That is, a change of one unit in the pH scale represents a 10-fold change in acidity or alkalinity. A soil with a pH of 5.0 is 10 times more acidic than a soil with a pH of 6.0 and 100 times more acidic than a soil with a pH of 7.0. This is one good reason to be very careful in trying to increase or lower soil pH. Some factors, including soil type and organic matter, affect the amount of materials needed to change soil pH. Fertilizers and rain affect soil pH. Organic matter and soil microorganisms are a few other factors that affect soil pH.

Agricultural limestone normally is used to increase the soil's pH. Sulfur is normally used to lower the soil pH. But fertilizer and water normally change the soil pH more rapidly. Plants have specific pH requirements for normal growth. Most ornamental plants grow well in a pH range of 6.0 to 7.0. But azaleas, gardenias, camellias, and

related plants require a soil pH level between 4.5 to 5.5. It is important to know the pH levels and nutrient requirements of your ornamental plants to ensure normal growth and success. The following table lists some woody and herbaceous landscape plants and their desired soil pH ranges.

1.1.9 Irrigation in Agriculture

When it comes to agriculture, irrigation usually comes down to applying controlled amounts of water to assist in the production of crops, as well as to grow landscape plants and lawns. There are several different methods of watering that can be applied depending on the type of terrain, crop, water source availability, reservoirs, available infrastructure, and so on. Due to climate change and the need for increased food production, the amount of irrigation water has drastically increased and will continue to increase.

Irrigated farming represents 20 percent of the total cultivated land, but contributes to 40 percent of the total food produced worldwide.

Different Ways to water plants With Irrigation Systems

Irrigation systems are widely used in every crop production in order to apply the amount of water needed for the crop. The practice is also known as irrigation management. Despite its broad application, irrigation should occur in a uniform and timely manner in order to minimize losses and damage to soil, water, air, plant, and animal resources.

Flood or furrow irrigation

entire soil surface is covered with water; it moves over the field by gravity flow

Sprinkler irrigation

crops are irrigated with high-pressure sprinklers set in the field; it can be solid or hand-moved

Drip irrigation

water is placed directly into the crop root zone from the low flow emitters, this usually also involves drip irrigation systems

Center Pivot irrigation

single central irrigation pipeline rotates around the pivot point. As it rotates, water sprinklers along the central pipe and irrigates crops

Traditional farming methods are often inefficient, with many farmers relying on guesswork and manual labor to manage their crops. This can lead to:

1.1.10 Overuse of water and fertilizers

Overuse of Water:

Water Scarcity: Depletion of groundwater resources, leading to reduced availability for future use.

Soil Salinization: Increased salt concentrations in soil, reducing fertility and affecting plant growth.

Water Pollution: Contamination of surface and groundwater with agricultural runoff, harming aquatic ecosystems.

Energy Waste: Inefficient use of energy for pumping and treating water.

Crop Damage: Overwatering can lead to root rot, nutrient deficiencies, and reduced crop yields.

Overuse of Fertilizers:

Soil Degradation: Excess nutrients can alter soil pH, reduce microbial activity, and increase erosion.

Water Pollution: Fertilizer runoff can contaminate waterways, harming aquatic life and human health.

Air Pollution: Volatilization of ammonia and other gases contributes to air pollution.

Economic Losses: Overfertilization can lead to reduced crop yields, lower quality produce, and increased costs.

Environmental Harm: Excess nutrients can stimulate harmful algal blooms, reduce biodiversity, and alter ecosystems.

Smart farming addresses these challenges by providing farmers with real-time data and insights to make informed decisions about their crops.

1.1.11 Need for smart farming arises from various factors, including:

Global food security: Increasing global population demands more food production.

Climate change: Rising temperatures, changing precipitation patterns, and increased frequency of extreme weather events impact agricultural productivity.

Water scarcity: Limited water resources necessitate efficient irrigation management.

Labor shortages: Aging farmer populations and urbanization lead to labor scarcity.

Environmental concerns: Sustainable practices are needed to reduce agriculture's ecological footprint.

Economic pressures: Farmers face fluctuating market prices, reducing profit margins.

Soil degradation: Intensive farming practices lead to soil erosion, nutrient depletion, and reduced fertility.

Food waste: Inefficient supply chains result in significant food waste.

Consumer demands: Increasing consumer awareness of food origin, quality, and sustainability drives demand for transparent and responsible farming practices.

1.2 Problem Statement

Develop an IoT-based smart farming system that utilizes sensors, data analytics, and automation to optimize crop yields, reduce water and chemical usage, and improve resource allocation, while minimizing environmental impact and enhancing decision-making capabilities for farmers, thereby increasing productivity, profitability, and sustainability of farming operations.

Design and implement a smart farming system using IoT sensors and AWS cloud services to optimize crop yields, reduce water consumption, and predict weather-related crop damage

1.2.1 The system should:

- 1. Collect real-time** data on soil moisture, temperature, and humidity from IoT sensors.
- 2. Analyze data** using AWS cloud services (AWS IoT Core, AWS Lambda, Amazon S3) to identify trends and patterns.
- 3. Provide personalized recommendations** to farmers on optimal irrigation schedules, fertilization, and pest control.
- 4. Predict weather-** for crop damage using machine learning algorithms and AWS services
- 5. Integrate** with existing farm management systems and mobile apps for seamless data exchange.
- 6. Ensure data security**, scalability, and reliability using AWS cloud services.

1.2.2 Goals:

1. Increase crop yields by 20% through data-driven decision making.
2. Reduce water consumption by 30% through optimized irrigation schedules.
3. Predict weather-related crop damage with 90% accuracy.
4. Improve farmer engagement and adoption of smart farming practices.

1.2.3 Constraints:

1. Limited internet connectivity in rural areas.
2. High upfront costs for IoT sensors and AWS services.
3. Data privacy and security concerns.
4. Integration with existing farm management systems and mobile apps."

This problem statement outlines the key challenges and goals for a smart farming system using IoT and AWS cloud services. It provides a clear direction for designing and implementing a solution that addresses the specific needs of farmers and the agricultural industry.

1.3 Background and context

The applications of Internet of Things in agriculture target conventional farming operations to meet the increasing demands and decrease production losses. IoT in agriculture uses robots, drones, remote sensors, and computer imaging combined with continuously progressing machine learning and analytical tools for monitoring crops, surveying, and mapping the fields, and provide data to farmers for rational farm management plans to save both time and money.

Agriculture implements IoT through use of robots, drones, sensors, and computer imaging integrated with analytical tools for getting insights and monitor the farms. Placement of physical equipment on farms monitors and records data, which is then used to get valuable insights.

As a result The advancement of science and technology, the presence of this reality has encouraged the development of smart farming, which use sensors and irrigation systems to manage crops as they grow. With sensor-based computer applications, more accurate information about the crop, soil, and environment may be gathered. It promotes high-quality process and raw materials throughout the entire product process. This is because utilising the Internet of Things in smarter agriculture makes it more competitive than traditional methods. Combined with IoT-based smart agriculture technologies, organic agricultural agriculture and family farming may see a benefit.

Sustainable use of water and input and treatment optimization will allow farmers to produce more food while also preserving the environment. To include agriculture in smart use of natural resources, usage of technologies such as remote control, decision support tools, automated irrigation systems, frost avoidance, and fertilisation is required. These activities are supplied by IoT technologies, which provide devices like as hardware, intelligent apps, integration platforms, control procedures, operating systems, and cloud computing.

The benefits of IoT and the Internet may be gained through the Cloud of Things, which combines IoT with cloud computing. Another requirement for the IoT is for it to provide society with information transparency. This work summarizes the current IoT-based agricultural tools and applications, which are broken down into distinct areas below.

1.3.1 Importance of Smart Farming

Increased Crop Yields: Smart Farming helps optimize crop growth, leading to increased yields and better quality produce. These smart farming techniques can lead to:

- 10-20% increase in crop yields
- Improved crop quality
- Reduced waste and losses

- Enhanced decision-making
- Increased profitability
- Sustainable agriculture practices

Water Conservation: Precision irrigation systems and soil moisture sensors reduce water waste and ensure optimal water usage. By implementing these water conservation strategies, farmers can:

- Reduce water waste
- Lower energy consumption
- Increase crop yields
- Improve water productivity
- Enhance sustainability

Reduced Chemical Usage: Targeted application of fertilizers and pesticides minimizes environmental impact and promotes sustainable practices. By adopting these smart farming practices, farmers can:

- Reduce chemical usage by 20-30%
- Minimize environmental impact
- Promote soil health and biodiversity
- Improve crop quality and yields
- Enhance sustainability and eco-friendliness

Improved Resource Allocation: Data-driven decisions optimize resource usage, reducing waste and improving efficiency.

By improving resource allocation, farmers can:

- Increase efficiency by 20-30%
- Reduce costs by 15-25%
- Enhance profitability by 10-20%
- Minimize environmental impact
- Promote sustainable agriculture practices

Enhanced Decision-Making: Real-time data and analytics enable farmers to make informed decisions, reducing uncertainty and improving outcomes.

By enhancing decision-making, farmers can:

- Improve crop yields by 10-20%
- Reduce costs by 15-25%
- Minimize environmental impact
- Increase efficiency by 20-30%
- Enhance profitability by 10-20%
- Promote sustainable agriculture practices

Increased Efficiency: Automation and optimized processes reduce labor costs and improve productivity.

By increasing efficiency, farmers can:

- Reduce costs by 15-25%

- Improve crop yields by 10-20%
- Enhance profitability by 10-20%
- Minimize environmental impact
- Promote sustainable agriculture practices
- Increase productivity by 20-30%

Better Supply Chain Management: Real-time tracking and monitoring ensure timely delivery of fresh produce to consumers.

By improving supply chain management, farmers and suppliers can:

- Reduce waste by 10-20%
- Increase efficiency by 15-25%
- Enhance profitability by 10-20%
- Improve quality and freshness
- Promote sustainability and transparency
- Strengthen relationships with suppliers and retailers

Environmental Sustainability: Smart Farming promotes eco-friendly practices, conserving natural resources and mitigating climate change.

By adopting environmentally sustainable practices, smart farming can:

- Reduce carbon footprint by 20-30%
- Conserve water by 15-25%
- Promote biodiversity and ecosystem services
- Enhance soil health and fertility
- Support climate change mitigation and adaptation
- Ensure a sustainable food future

Food Security: Increased crop yields and improved resource allocation help meet the world's growing food demands.

By addressing food security through smart farming, we can:

- Ensure global food availability by 2030
- Reduce hunger and malnutrition
- Improve food quality and safety
- Support sustainable agriculture practices
- Enhance climate resilience
- Promote equitable access to nutritious food

Economic Benefits: Smart Farming improves profitability for farmers, contributing to local economic growth and development.

By adopting smart farming practices, farmers and agricultural businesses can:

- Increase profitability by 10-20%
- Boost yields by 15-25%
- Reduce costs by 10-20%
- Create new business opportunities
- Gain a competitive advantage
- Contribute to rural development

By adopting Smart Farming practices, farmers can improve productivity, sustainability, and profitability, ultimately contributing to a more food-secure and environmentally conscious future.

1.4 Objectives

1.4.1 Use IOT Sensors for precision Agriculture

Optimize crop yields and reduce waste using data-driven insights.

Precision Agriculture (PA) is an approach to farming that uses advanced technology, sensors, and data analysis to optimize crop yields, reduce waste, and promote sustainability.

IOT sensor Collection of data includes

1.4.1.1 Soil Moisture Monitoring: Collect data on soil moisture levels to optimize irrigation schedules and reduce water waste.

1.4.1.2 Temperature and Humidity Monitoring: Collect data on temperature and humidity levels to monitor crop health and optimize growing conditions

1.4.1.3 Weather Monitoring: Collect data on weather conditions like rainfall, sunset, sunrise , heat index dew point to optimize farming practices and predict weather-related events.

1.4.1.4 Equipment Monitoring: Collect data on equipment performance and usage to optimize maintenance schedules and reduce downtime.

1.4.2 Use IOT devices to autonomous and remote control

Autonomous and remote control equipment is a key aspect of smart farming, enabling farmers to:

Remotely monitor and control equipment, reducing the need for physical presence.

Automate tasks, such as planting, spraying, and harvesting, increasing efficiency and accuracy.

Optimize equipment usage, reducing fuel consumption and wear and tear.

Improve safety, by minimizing human exposure to hazardous conditions.

1.4.3 Provide predictive and statical analysis from data

Statistical analysis that can be performed on data from IoT sensors

1.4.3.1 Soil Moisture Analysis:

- Average soil moisture level: 60%
- Standard deviation: 10%
- Correlation with irrigation schedules: 0.9

- Regression analysis: Soil moisture level = 0.8 x Irrigation schedule + 0.2 Weather data

1.4.3.2 weather data Analysis:

- Improved crop management
- Enhanced decision-making
- Reduced weather-related losses
- Optimized irrigation schedules
- Increased crop yields
- Better resource allocation
- Reduced environmental impact

1.5 Methodology

Smart farming is a management concept focused on providing the agricultural industry with the infrastructure to leverage advanced technology – including big data, the cloud and the internet of things (IoT) – for tracking, monitoring, automating and analyzing operations. Also known as precision agriculture, smart farming is software-managed and sensor-monitored. Smart farming is growing in importance due to the combination of the expanding global population, the increasing demand for higher crop yield, the need to use natural resources efficiently, the rising use and sophistication of information and communication technology and the increasing need for climate-smart agriculture.

By making farming more connected and intelligent, precision agriculture helps reduce overall costs and improve the quality and quantity of products, the sustainability of agriculture and the experience for the consumer. Increasing control over production leads to better cost management and waste reduction. The ability to trace anomalies in crop growth or livestock health, for instance, helps eliminate the risk of losing yields. Additionally, automation boosts efficiency. With smart devices, multiple processes can be activated at the same time, and automated services enhance product quality and volume by better controlling production processes.

Smart farming systems also enable careful management of the demand forecast and delivery of goods to market just in time to reduce waste. Precision agriculture is focused on managing the supply of land and, based on its condition, concentrating on the right growing parameters – for example, moisture, fertilizer or material content – to provide production for the right crop that is in demand. The types of precision farming systems implemented depend on the use of software for the management of the business. Control systems manage sensor input, delivering remote information for supply and decision support, in addition to the automation of machines and equipment for responding to emerging issues and production support.

Smart farming incorporates information and communication technologies into machinery, equipment and sensors used in agricultural production systems. Technologies such as the IoT and cloud computing are advancing this development even further by introducing more robots and artificial intelligence into farming.

For example, farmers can use smartphones and tablets to access real-time data about the condition of almost anything involved in their day-to-day operations:

Smart agricultural practices allow for the generation of a large volume of data and information. Farmers can use this information to make data-based decisions and take action for improved productivity and profitability.

1.5.1 What is the IoT?

The Internet of Things (IoT) refers to the network of physical devices, vehicles, buildings, and other items that are embedded with sensors, software, and connectivity, allowing them to collect and exchange data with other devices and systems over the internet.

These devices, also known as "smart devices," can collect and share data, enabling them to interact with the physical world and with each other. The IoT enables devices to be controlled remotely, and it provides a level of automation and efficiency that was previously unimaginable.

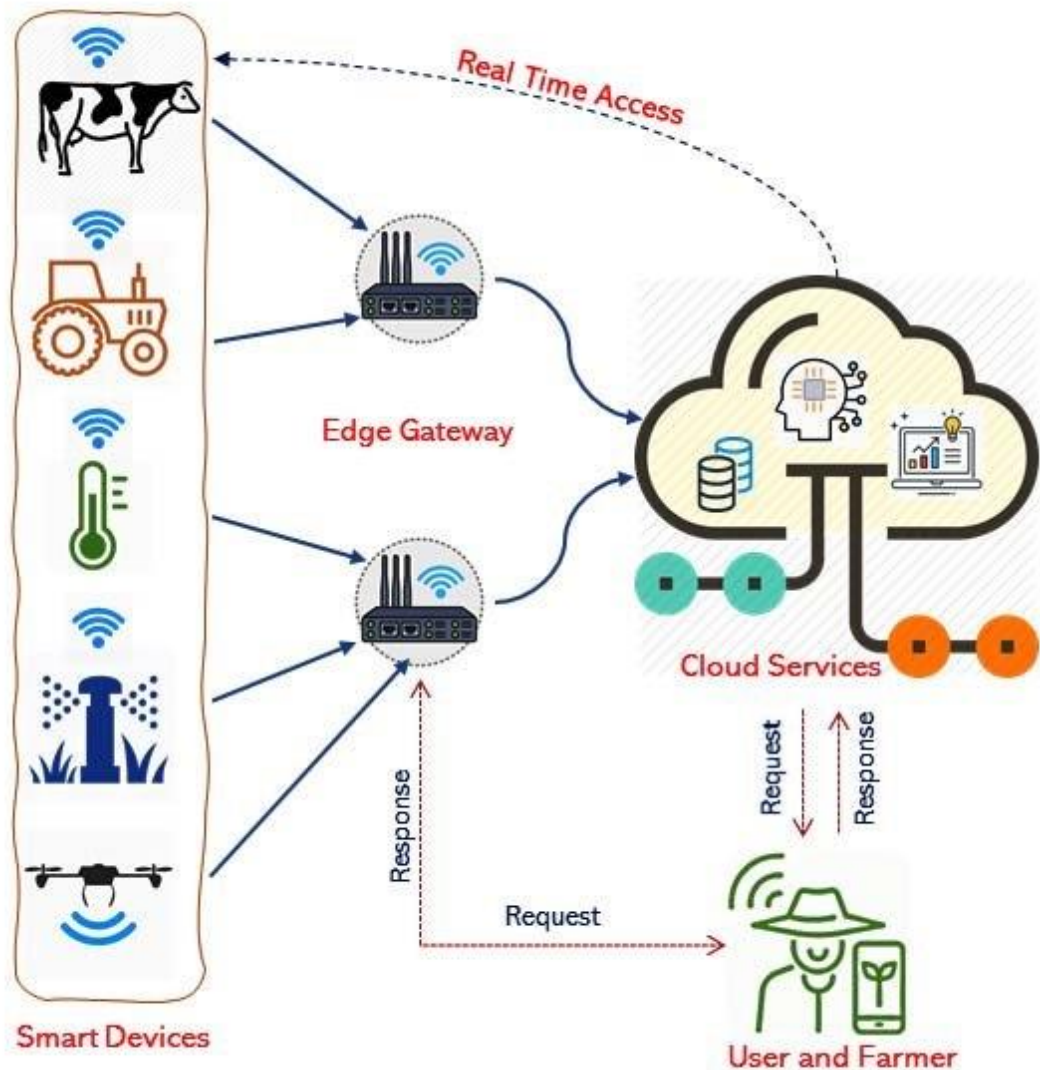
1.5.2 Examples of IoT devices:

1. Smart home devices (e.g., thermostats, lights, security cameras)
2. Wearable devices (e.g., smartwatches, fitness trackers)
3. Industrial sensors (e.g., temperature, pressure, vibration)
4. Autonomous vehicles
5. Smart city infrastructure (e.g., traffic management, energy grids)
6. Medical devices (e.g., heart rate monitors, insulin pumps)
7. Agricultural sensors (e.g., soil moisture, weather stations)

Allows objects to be controlled remotely via a network infrastructure.

It creates opportunities for direct integration between the physical world and digital systems. In this guide, we discuss how the IoT is used in agriculture.

1.6 Smart Farming Infrastructure



Smart Farming infrastructure can be further divided in to different components

1.6.1 Hardware Components

1.6.1.1 Sensors

- Soil moisture sensors
- Temperature and humidity sensors
- Light sensors
- Weather data collector

1.6.1.2 Microcontrollers:

- Arduino or Raspberry Pi boards

1.6.1.3 Communication Modules

- Wi-Fi or cellular modules (e.g., ESP8266/ESP32 or SIM900)

1.6.1.4. Actuators

- Water pumps
- Valves
- LED grow lights

1.6.1.5 Power Supply

- Solar panels or batteries

1.6.2 Software Components:

1.6.2.1 Operating System

- Linux or Windows IoT, RTOS

1.6.2.2 Programming Language

- Python, C++, or Java ,Embedded C

1.6.2.3 IoT Platform:

- AWS IoT, Google Cloud IoT Core, or Microsoft Azure IoT Hub

1.6.2.4 Data Analytics:

- Data visualization tools (e.g., Tableau or Power BI), Graph UI

1.6.3 System Design:

1.6.3.1 Sensor Node:

- Connect sensors to microcontrollers
- Send data to IoT platform via communication modules

1.6.3.2. Gateway:

- Receive data from sensor nodes
- Send data to IoT platform

1.6.3.3 IoT Platform:

- Receive data from gateways
- Analyze data using machine learning algorithms
- Send commands to actuators via gateways

1.6.3.4 Actuator Node:

- Receive commands from IoT platform
- Control actuators (e.g., water pumps or LED grow lights)

1.6.4 Smart Farming Applications:

1. Precision Irrigation:

- Monitor soil moisture levels
- Automate watering schedules

2. Crop Monitoring:

- Monitor temperature, humidity, and light levels
- Detect anomalies and alert farmers

3. Automated Farming:

- Automate tasks (e.g., pruning or harvesting)
- Optimize farming operations

Chapter 2

2.1 Literature Review

Smart farming, also known as precision agriculture, is an innovative approach to farming that uses advanced technology to optimize crop yields, reduce waste, and promote sustainability.

2.1.1 Early Studies

- "Precision Agriculture: A Review" (2001) by Robert et al. - One of the first comprehensive reviews of precision agriculture, highlighting its potential to improve efficiency and reduce environmental impact.
- "Agricultural Robotics and Automation" (2007) by Billingsley et al. - Explores the use of robotics and automation in agriculture, including autonomous tractors and crop monitoring systems.

2.1.2 Recent Advances

- "Smart Farming: A Review of the Literature" (2020) by Kamilaris et al. - A comprehensive review of smart farming technologies, including IoT, AI, and big data analytics.
- "Precision Agriculture 2.0: A Review of the Latest Advances" (2022) by Zhang et al. - Discusses recent advancements in precision agriculture, including the use of drones, satellite imaging, and machine learning.

2.1.3 Key Technologies

- "IoT in Agriculture: A Review" (2019) by Ray et al. - Examines the use of IoT sensors and devices in agriculture, including soil moisture monitoring and livestock tracking.
- "Machine Learning in Agriculture: A Review" (2020) by Liakos et al. - Reviews the application of machine learning algorithms in agriculture, including crop yield prediction and disease detection.

2.1.4 Case Studies

- "Smart Farming in the USA: A Case Study" (2019) by USDA - Presents a case study of smart farming adoption in the United States, highlighting benefits and challenges.
- "Precision Agriculture in Europe: A Review of Case Studies" (2020) by EU Commission - Summarizes several case studies of precision agriculture adoption in Europe, showcasing successful implementations.

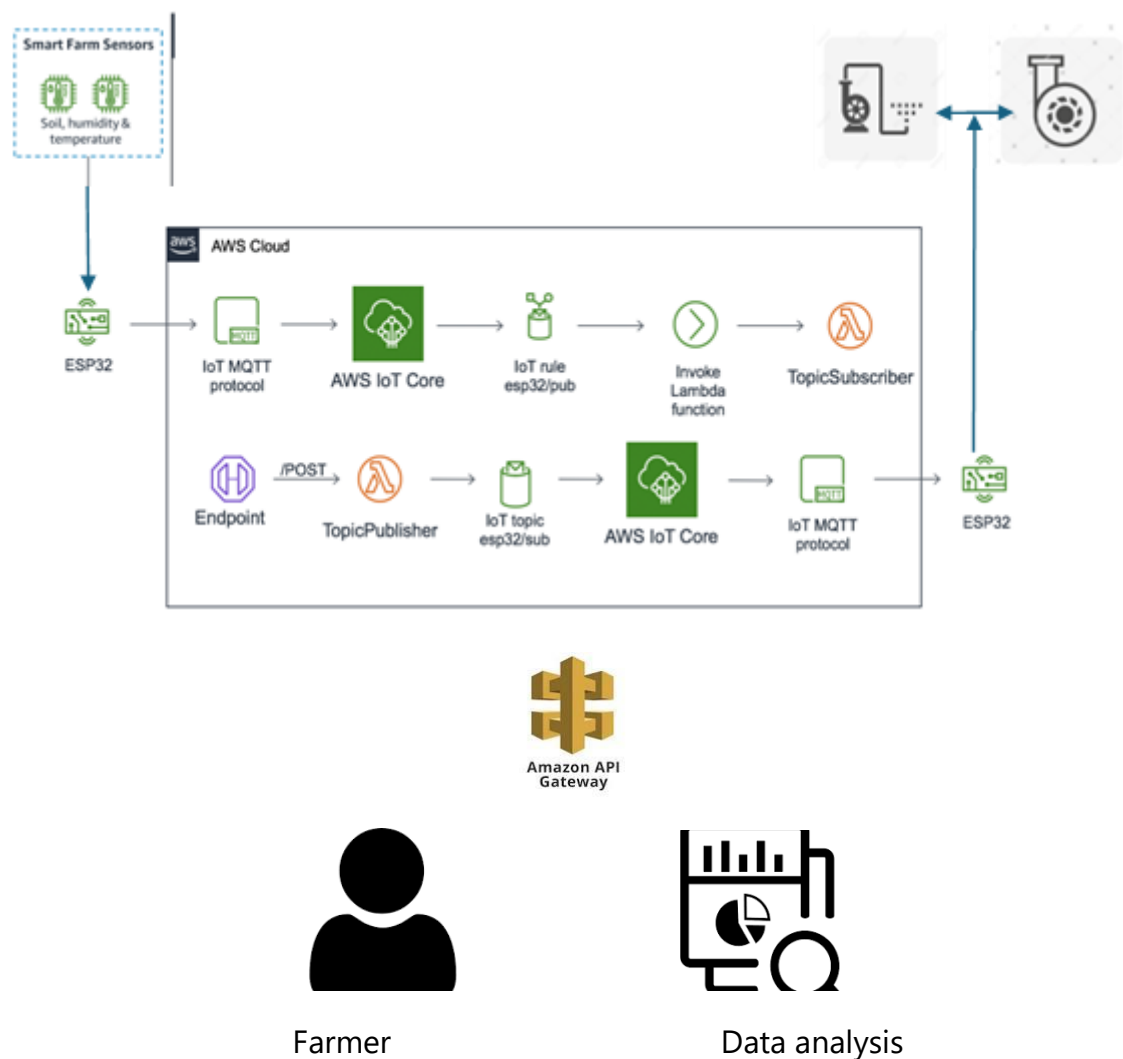
2.1.5 Challenges and Future Directions

- "Challenges in Smart Farming: A Review" (2020) by Kumar et al. - Discusses challenges in smart farming adoption, including data management, cybersecurity, and scalability.
- "Future Directions in Smart Farming: A Review" (2022) by Chen et al. - Explores future directions in smart farming, including the integration of AI, robotics, and blockchain technology.

2.2 Proposed Method

Smart farming is an innovative approach to agriculture that leverages technology to enhance crop yields, reduce waste, and promote sustainable practices. This report outlines a smart farming project that utilizes Amazon Web Services (AWS) and an ESP32 microcontroller to monitor and control farm conditions remotely.

2.2.1 System Design:



2.2.2 System Components

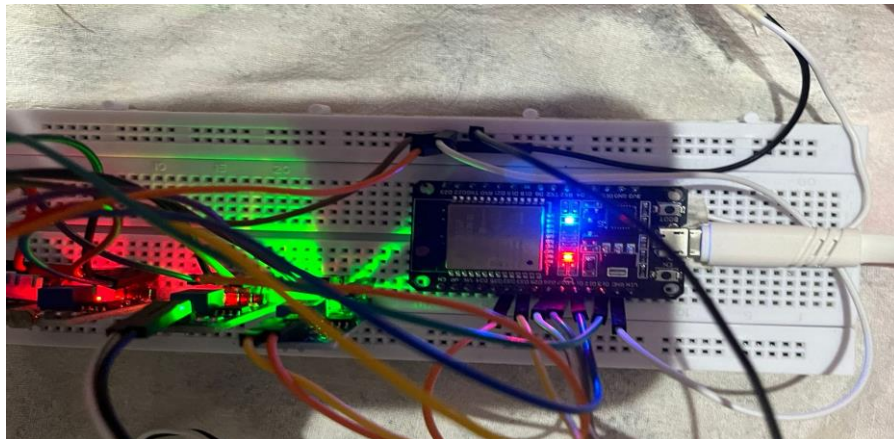
2.2.2.1 Hardware

- **ESP32 Microcontroller:**
- **Sensors**
 - **DHT111 Temperature and Humidity sensor**
 - **Soil Moisture sensor**
 - **Light Sensor**
 - **Rain Fall Sensor**
 - **Relay board for motor control**

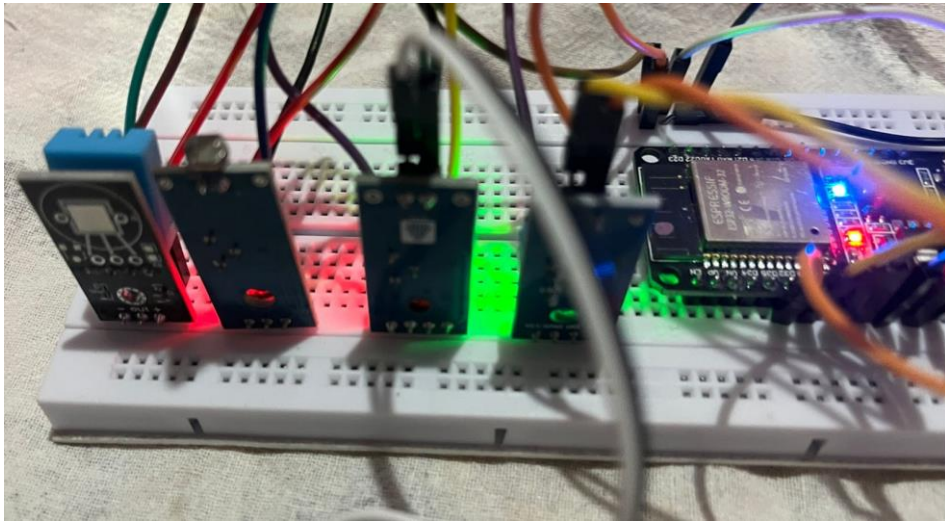
2.2.2.2 Software

- **ESP32 Firmware**
- **AWS cloud services**
- **Graph UI for Statical analysis**

2.2.2.3 Proof of concept design



ESP32



Rainfall, Temperature, Light and Moisture sensors

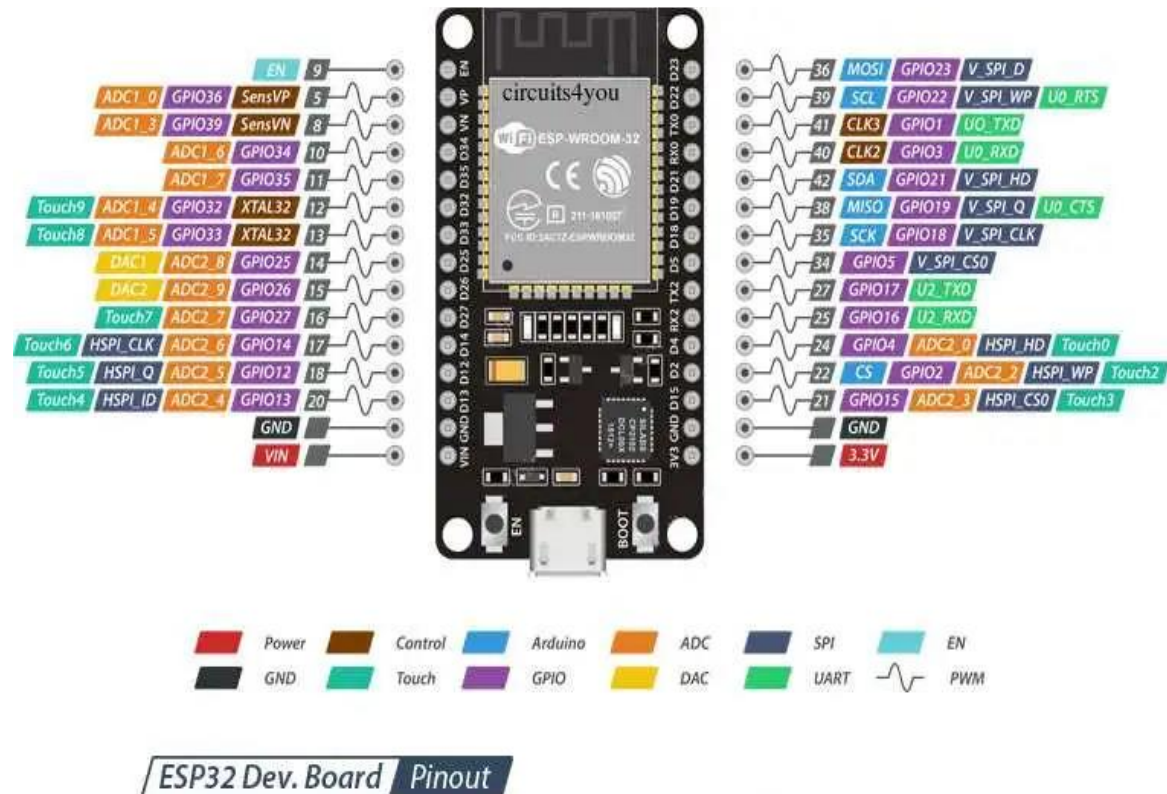


Sensor Probes

Relay Board

2.3 Hardware

2.3.1 ESP32 Microcontroller



The ESP32 is suitable for various applications, such as:

IoT Devices: Smart home automation, industrial sensors, and wearables.

Robotics: Autonomous robots, robotic arms, and robotic vehicles.

Wearables: Smartwatches, fitness trackers, and health monitors.

Smart Cities: Environmental monitoring, traffic management, and public safety.

Agriculture: Smart farming, soil monitoring, and livestock tracking.

Advantages

Affordability: Low-cost compared to other microcontrollers.

Ease of Use: Simple programming and development environment.

Flexibility: Supports multiple programming languages and applications.

Community Support: Large community of developers and resources available.

Limitations

Limited Memory: 520 KB SRAM and 4 MB flash memory.

Complexity: Steeper learning curve due to its advanced features.

2.3.2 Sensors

2.3.2.1 Soil Moisture Sensor:

A soil moisture sensor is a device that measures the moisture levels in soil. It's a crucial tool in smart farming, gardening, and environmental monitoring. Here's a detailed overview

Types:

1. Resistance-based sensors
2. Capacitance-based sensors
3. Tensiometers
4. Time-domain reflectometry (TDR) sensors
5. Frequency-domain reflectometry (FDR) sensors

Working Principle:

- 1. Resistance-based sensors:** Measure the resistance between two electrodes inserted into the soil. Moist soil conducts electricity better than dry soil.
- 2. Capacitance-based sensors:** Measure the changes in capacitance between two electrodes inserted into the soil. Moist soil has a higher capacitance than dry soil.
- 3. Tensiometers:** Measure the soil water tension, which indicates the energy required to extract water from the soil.
- 4. TDR/FDR sensors:** Measure the time/frequency domain reflectometry of electromagnetic pulses inserted into the soil, which indicates the soil moisture levels.

Applications:

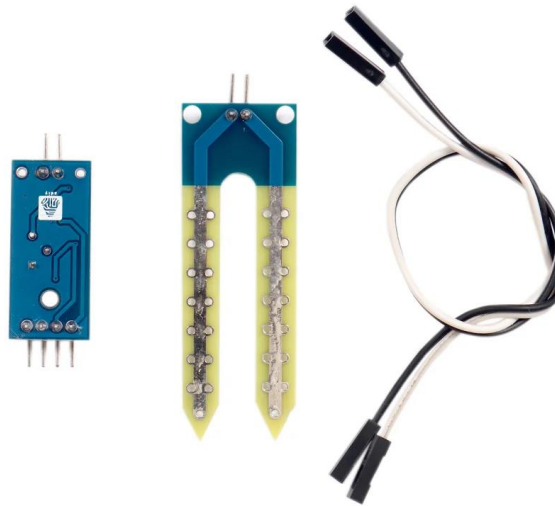
- 1. Smart farming:** Optimize irrigation, reduce water waste, and improve crop yields.
- 2. Gardening:** Monitor soil moisture levels for optimal plant growth.
- 3. Environmental monitoring:** Track soil moisture levels for research, conservation, and ecology studies.
- 4. Landscaping:** Monitor soil moisture levels for optimal turf and plant growth.

Advantages:

1. Real-time monitoring
2. Accurate measurements
3. Low maintenance
4. Cost-effective

Limitations:

1. Calibration required
2. Soil type and temperature affect accuracy
3. Limited depth measurement (for some sensors)



Description

The Soil Moisture Sensor is a simple breakout for measuring the moisture in the soil and similar materials. The soil moisture sensor is pretty straightforward to use. The two large, exposed pads function as probes for the sensor, together acting as a variable resistor. The more water that is in the soil means the better the conductivity between the pads will be and will result in lower resistance, and a higher SIG out.

To get the Soil Moisture Sensor functioning all you will need is to connect the VCC and GND pins to your Arduino-based device (or compatible development board) and you will receive a SIG out which will depend on the amount of water in the soil.

Features:

- Soil Moisture Sensor

- Simple to use

- Passive component

- Resistive Sensor

Specifications:

- Operating voltage: 3.3V~5V

Dual output mode, analog output more accurate

A fixed bolt hole for easy installation

With power indicator (red) and digital switching output indicator (green)

Having LM393 comparator chip, stable

Panel PCB Dimension: Approx. 3cm x 1.5cm

Soil Probe Dimension: Approx. 6cm x 3cm

Cable Length: Approx. 21cm

VCC: 3.3V-5V

GND: GND

DO: digital output interface(0 and 1)

AO: analog output interface

How Does a Soil Moisture Sensor Work?

The soil moisture sensor operates in a straightforward manner.

The fork-shaped probe with two exposed conductors acts as a variable resistor (similar to a potentiometer) whose resistance varies with the soil's moisture content.

This resistance varies inversely with soil moisture:

The more water in the soil, the better the conductivity and the lower the resistance.

The less water in the soil, the lower the conductivity and thus the higher the resistance.

The sensor produces an output voltage according to the resistance, which by measuring we can determine the soil moisture level.

2.3.2.2 ESP32 code for Moisture sensor

```
const int sensor_pin = A1;    /* Soil moisture sensor O/P pin */ void
setup() {
  Serial.begin(9600);         /* Define baud rate for serial communication */
}
void loop()
{
  float moisture_percentage;
  int sensor_analog;
  sensor_analog = analogRead(sensor_pin);
  moisture_percentage = ( 100-((sensor_analog/1023.00) *
  100 ) );
  Serial.print("Moisture Percentage = ");
  Serial.print(moisture_percentage); Serial.print("%\n\n");
  delay(1000);
}
```

2.3.2.3 Temperature and Humidity Sensor:

A temperature and humidity sensor is a device that measures the temperature and humidity levels in the air. It's a crucial tool in various applications, including:

Applications:

1. Weather monitoring
2. Indoor air quality control
3. Greenhouse management
4. Industrial process control
5. Smart homes and buildings
6. Agricultural monitoring
7. Medical devices

Types:

1. Digital temperature and humidity sensors (e.g., DHT11, DHT22)
2. Analog temperature and humidity sensors (e.g., LM35, HIH6130)
3. Infrared temperature sensors (e.g., MLX90614)
4. Capacitive humidity sensors (e.g., HC-SR04)

Working Principle:

Digital sensors: Use electrical resistance or capacitance changes to measure temperature and humidity.

Analog sensors: Use thermistors or thermocouples to measure temperature, and resistive or capacitive elements to measure humidity.

Infrared sensors: Measure thermal radiation to determine temperature.

Specifications:

Temperature range: -40°C to 125°C (typical)

Humidity range: 0-100% RH (typical)

Accuracy: $\pm 1-5\%$ (typical)

Resolution: 0.1-1°C and 0.1-1% RH (typical)

Interface: Digital (I2C, SPI, etc.) or analog output

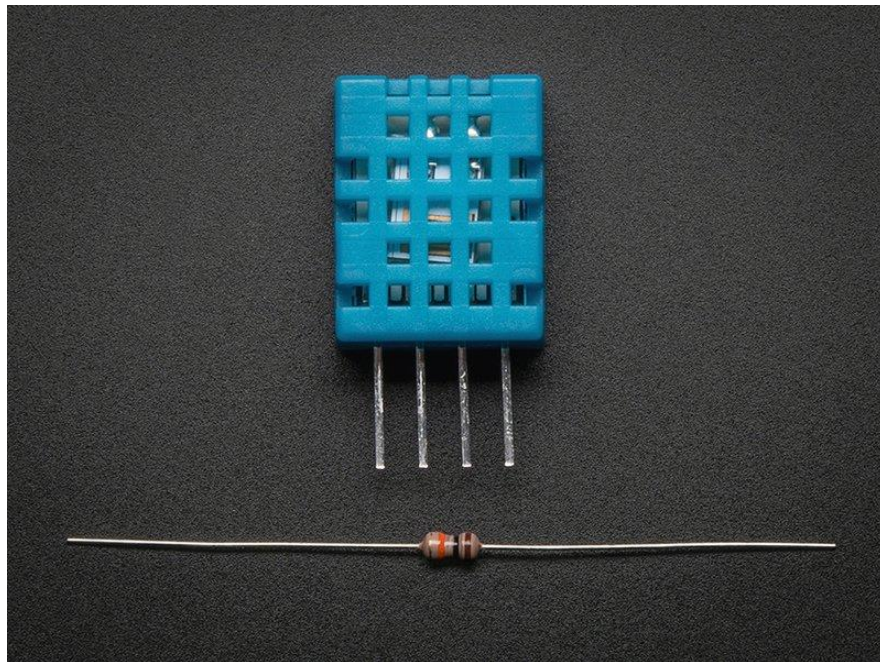
Advantages:

1. High accuracy and resolution
2. Low power consumption
3. Small size and low cost
4. Easy integration with microcontrollers

Limitations:

1. Calibration required
2. Sensitivity to environmental factors (e.g., dust, pollution)
3. Limited range and accuracy in extreme conditions

DHT11 temperature-humidity sensor



The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but

requires careful timing to grab data. You can get new data from it once every 2 seconds, so when using the library from Adafruit, sensor readings can be up to 2 seconds old.

Comes with a 4.7K or 10K resistor, which you will want to use as a pullup from the data pin to VCC.

Specifications:

- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20-80% humidity readings with 5% accuracy
- Good for 0-50 °C temperature readings +-2 °C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

Humidity is the measure of water vapour present in the air. The level of humidity in air affects various physical, chemical and biological processes. In industrial applications, humidity can affect the business cost of the products, health and safety of the employees. So, in semiconductor industries and control system industries measurement of humidity is very important.

Humidity measurement determines the amount of moisture present in the gas that can be a mixture of water vapour, nitrogen, argon or pure gas etc... Humidity sensors are of two types based on their measurement units. They are a relative humidity sensor and Absolute humidity sensor. DHT11 is a digital temperature and humidity sensor.

What is a DHT11 Sensor?

DHT11 is a low-cost digital sensor for sensing temperature and humidity. This sensor can be easily interfaced with any micro-controller such as Arduino, Raspberry Pi etc... to measure humidity and temperature instantaneously.

DHT11 humidity and temperature sensor is available as a sensor and as a module. The difference between this sensor and module is the pull-up resistor and a power-on LED. DHT11 is a relative humidity sensor. To measure the surrounding air this sensor uses a thermistor and a capacitive humidity sensor.

Working Principle of DHT11 Sensor

DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two

electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form.

For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. To get larger resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

The temperature range of DHT11 is from 0 to 50 degree Celsius with a 2-degree accuracy. Humidity range of this sensor is from 20 to 80% with 5% accuracy. The sampling rate of this sensor is 1Hz .i.e. it gives one reading for every second. DHT11 is small in size with operating voltage from 3 to 5 volts. The maximum current used while measuring is 2.5mA.

DHT11 sensor has four pins- VCC, GND, Data Pin and a not connected pin. A pull-up resistor of 5k to 10k ohms is provided for communication between sensor and micro-controller.

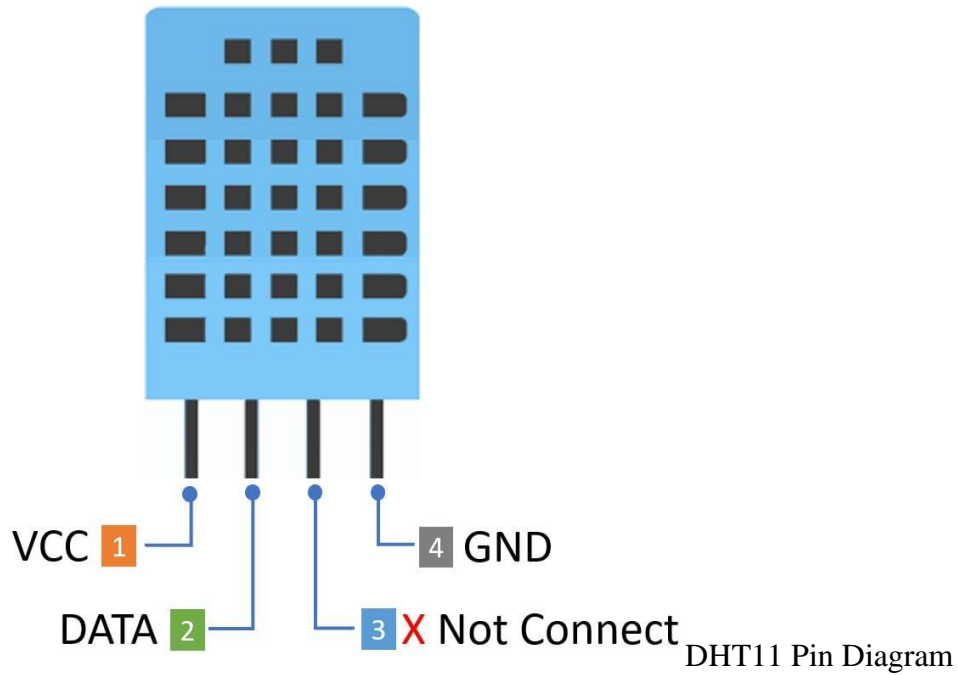
Applications

This sensor is used in various applications such as measuring humidity and temperature values in heating, ventilation and air conditioning systems. Weather stations also use these sensors to predict weather conditions. The humidity sensor is used as a preventive measure in homes where people are affected by humidity. Offices, cars, museums, greenhouses and industries use this sensor for measuring humidity values and as a safety measure.

- DHT11 sensor measures and provides humidity and temperature values serially over a single wire.
- It can measure the relative humidity in percentage (20 to 90% RH) and temperature in degrees Celsius in the range of 0 to 50°C.
- It has 4 pins; one of which is used for data communication in serial form.
- Pulses of different TON and TOFF are decoded as logic 1 or logic 0 or start pulse or end of a frame

DHT11 Pinout

- DHT11 is a 4-pin sensor, these pins are VCC, DATA, and GND and one pin is not in use shown in fig below.



DHT11 Pin Details and Electrical Characteristics

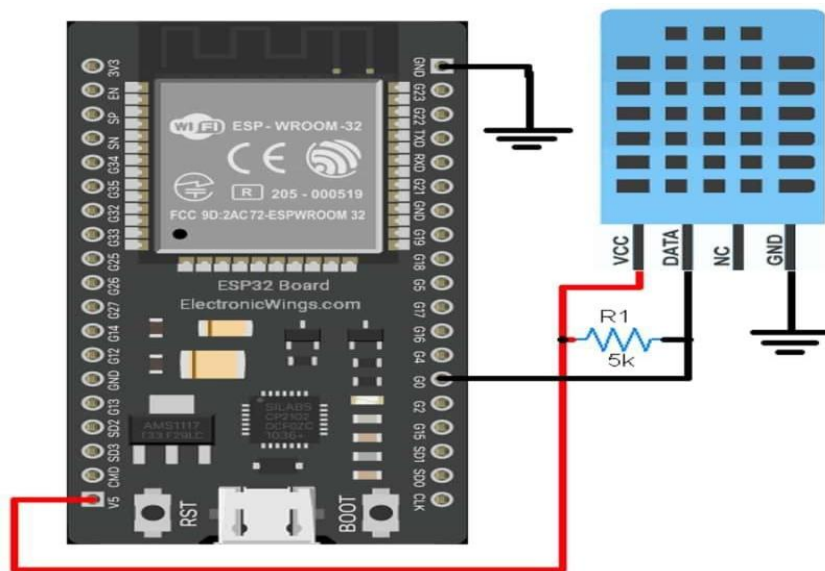
VCC: Power supply pin connects to 3 to 5.5V DC.

DATA: Output in digital pin

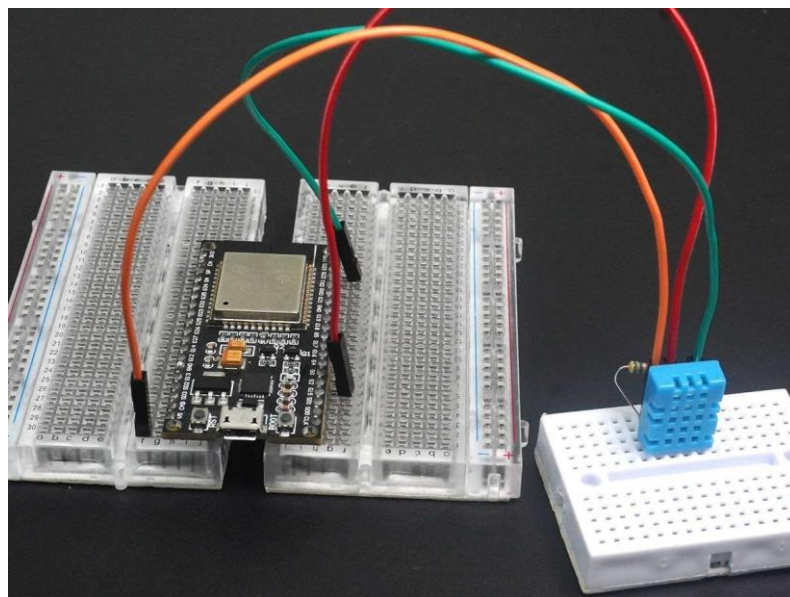
GND: Ground

Supply Current: 0.5mA to 2.5mA

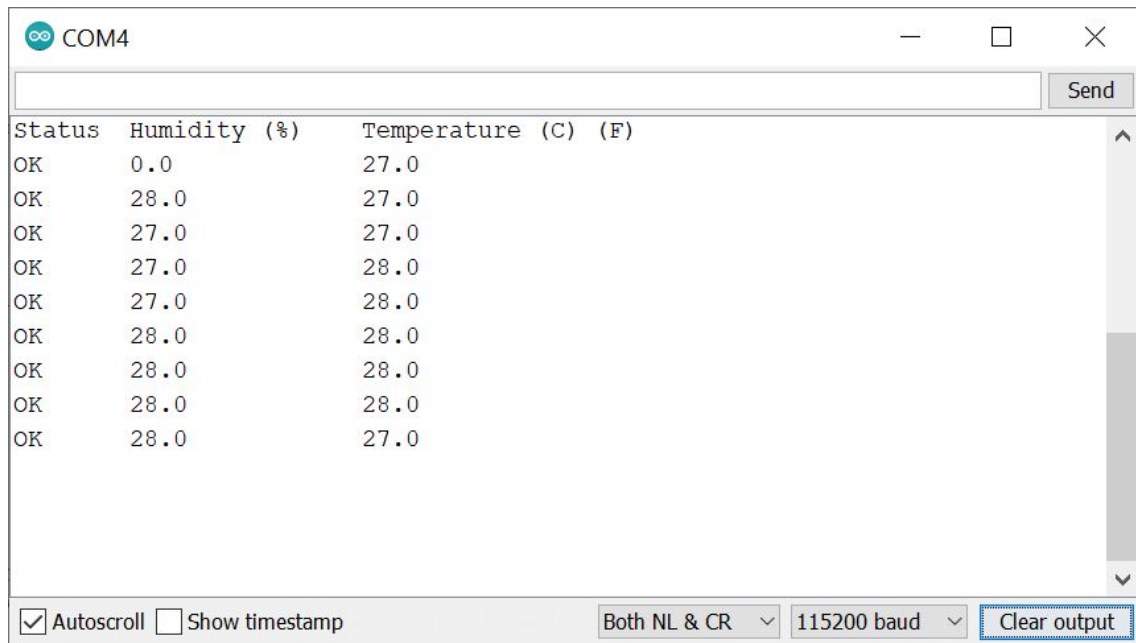
Standby Current: 100uA 150uA **Connection Diagram of DHT11 with ESP32**



ESP32 interfaced with DHT11



Reading DHT11 Sensor using ESP32



3.2.2.4 ESP32 code for DHT11 Sensor Interfacing

```
#include "DHT.h"
```

```
DHT dht;
```

```
float Humidity, Temperature;
```

```
void setup(void)
```

```
{  
    Serial.begin(115200); /*Set the baudrate to 115200*/  
    dht.setup(0); /*Connect the DHT11's data pin to GPIO0*/  
    Serial.println("Status\tHumidity (%)\tTemperature (C)\t(F)");  
    delay(1000); /*Wait for 1000mS*/  
}
```

```
void loop(void)
```

```
{  
    Temperature = dht.getTemperature(); /*Get the Temperature value*/  
    Humidity = dht.getHumidity(); /*Get the Humidity value*/  
    Serial.print(dht.getStatusString()); /*Get the Status of DHT11 Sensor*/  
    Serial.print("\t");  
    Serial.print(Humidity, 1);
```

```

Serial.print("\t\t");
Serial.println(Temperature, 1);
delay(dht.getMinimumSamplingPeriod()*2);
}

```

Now let's understand how code works.

we read the value Temperature and Humidity using following functions.

```

Temperature = dht.getTemperature();
Humidity = dht.getHumidity();

```

Now we tested the status of the DHT11 communication, which provides “OK” response on successful communication. If response “TIMEOUT” when, there is no response from DHT11.

```

Serial.print(dht.getStatusString());

```

Now we have printed the DHT11 values as follow:

We used “/t” is “Tab” horizontal space for proper presentation.

```

Serial.print("\t");
Serial.print(Humidity, 1);
Serial.print("\t\t");
Serial.println(Temperature, 1);

```

Now we have to provide a certain delay time interval before taking the next reading. There are two types of DHT sensors i.e., DHT11 and DHT22. DHT11 has 1 second time interval (1Hz sampling) for every reading and DHT22 has the 2 Second (0.5Hz Sampling). Accordingly, the function provides a time delay.

```

delay(dht.getMinimumSamplingPeriod()*2);

```

2.3.2.5 Light Intensity Sensor:

A light intensity sensor measures the amount of light present in an environment. It's a crucial tool in various applications, including:

Applications:

1. Smart lighting systems
2. Solar panels and renewable energy
3. Greenhouses and agriculture
4. Photography and cinematography
5. Indoor and outdoor lighting control
6. Automotive and aerospace industries
7. Medical and healthcare applications

Types:

1. Photodiodes
2. Phototransistors
3. Light-Dependent Resistors (LDRs)
4. Photomultiplier Tubes (PMTs)
5. Charge-Coupled Devices (CCDs)

Working Principle:

1. Photodiodes: Convert light into electrical current
2. Phototransistors: Amplify light-induced current
3. LDRs: Change resistance in response to light
4. PMTs: Amplify light-induced electrons
5. CCDs: Capture and measure light intensity

Specifications:

1. Sensitivity: Measures the minimum detectable light level
2. Range: Measures the maximum detectable light level
3. Resolution: Measures the smallest change in light intensity detectable

4. Spectral range: Measures the range of wavelengths detectable
5. Interface: Analog or digital output

Advantages:

1. High sensitivity and accuracy
2. Fast response time
3. Low power consumption
4. Small size and low cost
5. Easy integration with microcontrollers

Limitations:

1. Limited dynamic range
2. Sensitivity to temperature and noise
3. Limited spectral range
4. Calibration required

BH1750FVI light sensor



This is a Light Intensity Detection Module based on the BH1750 chip. BH1750FVI is a digital Ambient Light Sensor IC for the I2C bus interface. This IC is the most suitable to receive the ambient light data for adjusting LCD and Keypad backlight power of the Mobile phone. It is possible to detect wide range at High resolution.

The BH1750FVI is a digital light sensor that measures ambient light intensity. It's a popular choice for various applications, including:

- Smart lighting systems
- Display backlight control
- Solar panels and renewable energy
- Greenhouses and agriculture
- IoT devices and wearables

Key Features:

- High accuracy and sensitivity
- Digital output (I2C interface)
- Low power consumption (typical 0.65 mA)
- Wide measurement range (1-65535 lux)
- Fast response time (typical 2.5 ms)
- Small package (6-pin LGA)

How it Works:

1. The BH1750FVI uses a photodiode to convert light into electrical current.
2. The current is then amplified and converted into a digital signal.
3. The digital signal is output through the I2C interface.

Specifications:

- Measurement range: 1-65535 lux
- Accuracy: $\pm 20\%$
- Sensitivity: 1 lux
- Response time: 2.5 ms
- Power consumption: 0.65 mA (typical)
- Interface: I2C (6-pin LGA package)

Advantages:

- High accuracy and sensitivity
- Low power consumption
- Fast response time
- Small package size
- Easy integration with microcontrollers

Limitations:

- Limited dynamic range
- Sensitivity to temperature and noise
- Calibration required

Features:

No ambient light distinction
High precision determination accurate to 1 Lu for different lights
ROHM original package BH1750FVI chip

3.2.3.6 ESP32 interface code for light sensor

```
#include <BH1750.h>

#include <Wire.h>

BH1750 lightMeter;

void setup()

{

    Serial.begin(9600);

    // Initialize the I2C bus (BH1750 library doesn't do this automatically)

    Wire.begin();

    lightMeter.begin(BH1750::ONE_TIME_HIGH_RES_MODE);

    Serial.println(F("BH1750 One-Time Test"));

}

void loop()

{

    while (!lightMeter.measurementReady(true))

    {

        yield();

    }

    float lux = lightMeter.readLightLevel();

    Serial.print("Light: ");

    Serial.print(lux);

    Serial.println(" lx");

    lightMeter.configure(BH1750::ONE_TIME_HIGH_RES_MODE);

}
```

2.3.2.7 Rain Fall Sensor

A rainfall sensor is a device that measures the amount of rainfall over a specific period. It's commonly used in:

- Weather monitoring systems
- Agricultural automation
- Hydrological research
- Flood warning systems
- Smart cities and IoT applications

Types:

1. Tipping bucket rain gauge
2. Capacitance rain sensor
3. Resistance rain sensor
4. Ultrasonic rain sensor
5. Radar rain sensor

Working Principle:

1. Tipping bucket: Measures rainfall by counting the number of times a small bucket tips due to rainwater accumulation.
2. Capacitance: Measures changes in capacitance caused by rainwater accumulation.
3. Resistance: Measures changes in resistance caused by rainwater accumulation.
4. Ultrasonic: Measures the time-of-flight of ultrasonic pulses to detect rainwater accumulation.
5. Radar: Uses radar waves to detect and measure rainfall.

Specifications:

- Measurement range: 0-100 mm/h (typical)
- Accuracy: $\pm 5\%$ (typical)
- Resolution: 0.1 mm (typical)
- Response time: 1-10 seconds (typical)

- Interface: Analog or digital output

Advantages:

- High accuracy and resolution
- Fast response time
- Low power consumption
- Robust and weather-resistant design

Limitations:

- Calibration required
- Sensitivity to wind, temperature, and humidity
- Limited measurement range

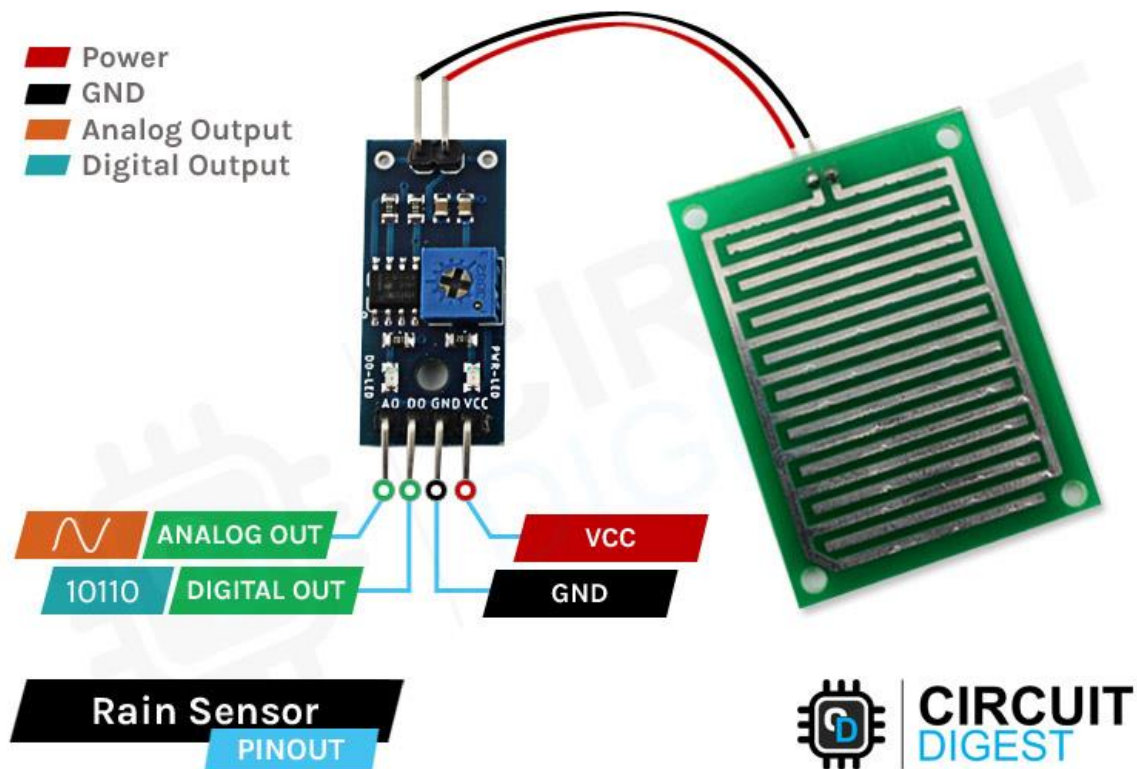
By using rainfall sensors, you can monitor and measure rainfall accurately, enabling applications such as weather forecasting, agricultural automation, and flood warning systems.

Rain Fall Sensor with ESP32

When it comes to detecting certain changes in weather conditions, it's pretty unpredictable, and rainfall is one of the most unexpected parameters of weather. So in this article, we decided to Interface the Rain Sensor with an Arduino. The raindrop sensor also known as rain detector sensor is an easy-to-use device that can detect rainfall. It acts as a switch when raindrops fall on the sensor, other than that with slight tweaks in the code, it can also measure the intensity of the rainfall. This sensor also has a separate indicator led and an onboard potentiometer through which you can adjust the sensitivity of the output digital signal provided by the sensor.

Rain Detection Sensor Pinout

Like Soil Moisture Sensor, the Rain detection sensor module also has four pins VCC, GND, Aout, and Dout that can be used to get the needful information out of the sensor, The pinout of the Rain Detection Sensor is given below:



VCC is the power supply pin of the Rain Detection Sensor that can be connected to 3.3V or 5V of the supply. But do note that the analog output will vary depending upon the provided supply voltage.

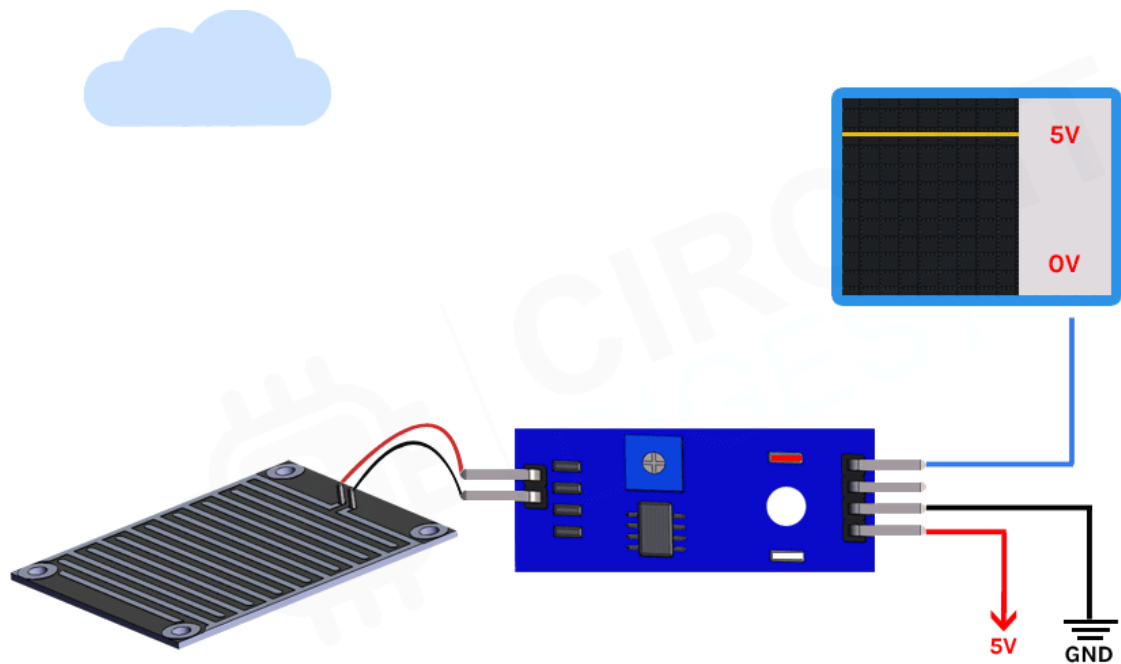
GND is the ground pin of the board and it should be connected to the ground pin of the Arduino.

DO is the Digital output pin of the board, output low indicates rain is detected, and high indicates no rain condition.

AO is the Analog output pin of the board that will give us an analog signal in between vcc and ground.

How does a Rain Detection Sensor Works

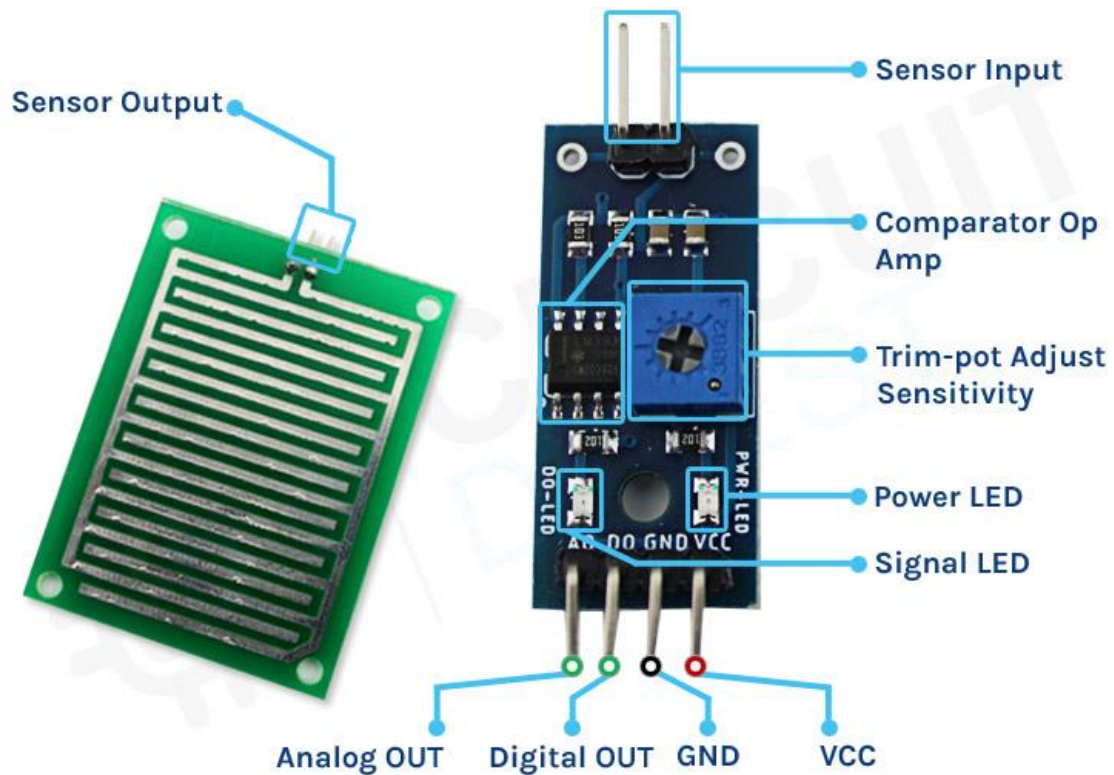
The working principle of the Rain Detection Sensor is pretty simple, as you can see in the image below. The PCB is made out of multiple exposed log conductive plates arranged in a grid format. When rain falls on top of the sensor the resistivity of the conductive plates changes, and by measuring the changes in the resistance, we can determine the intensity of the rainfall. The more intense the rainfall the lower the resistance.



The above image of the Rain Detection Sensor shows how the analog output of the sensor changes based on the water droplets falling on top of it. As you can see in the animation above the voltage slowly drops from VCC to 0V when water droplets start falling from the top, you can also see that the trigger LED on the board turns on when a certain threshold is reached which can be set by the potentiometer. For the sake of simplicity, we did not make any visual representation of how the digital part of the circuit works, but it's pretty simple when a certain threshold is reached (that we can set by a potentiometer) the output of the rain detection sensor module goes low and that lights up the onboard trigger LED.

Rain Detection Sensor - Parts

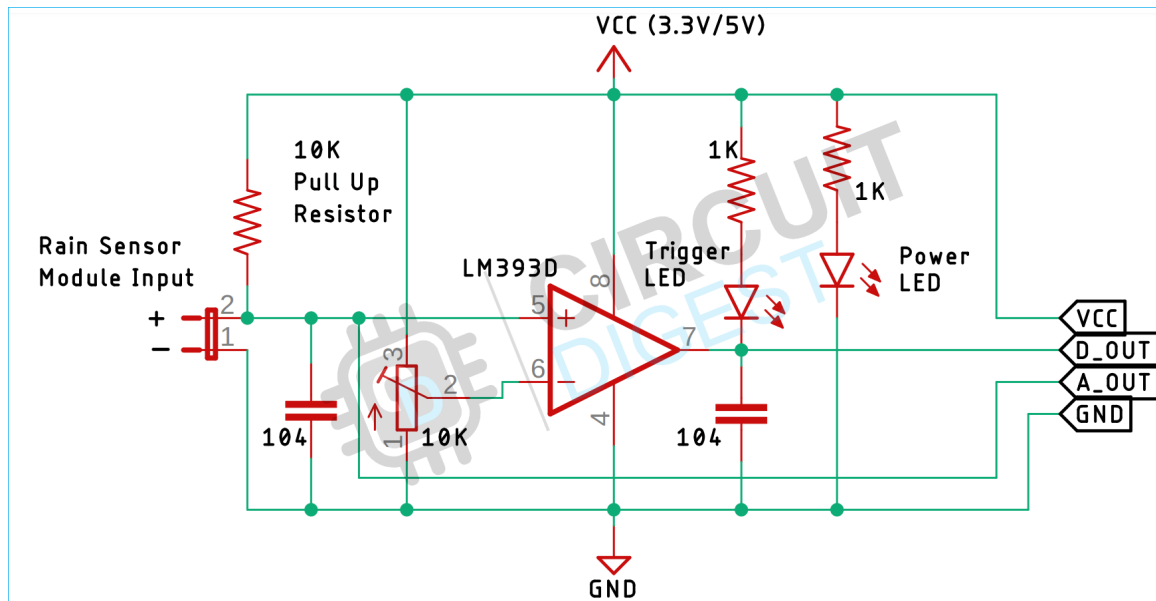
The entire rain detection sensor module consists of two parts: the rain detection sensor PCB and the signal processing module. The module processes the incoming data from the sensor PCB and it can output both analog and digital data simultaneously.



The sensor module has four pins, two of which are for VCC and Gnd and the other two can simultaneously output analog and digital data. As you can see in the image above the module has two onboard LEDs. The power led turns on when power is applied to the board and another one turns on when the set value by the potentiometer is reached. This board also has a comparator OP-Amp onboard that is responsible for converting the incoming analog signal from the photodiode to a digital signal. We also have a sensitivity adjustment potentiometer, with that, we can adjust the sensitivity of the device. And finally, we have the rain detection PCB together with the signal processing module that makes the rain detection sensor.

Rain Detection Sensor Module Circuit Diagram

The schematic diagram for the Rain Detection Sensor module is shown below. The schematic itself is very simple and needs a handful of generic components to build. If you don't have a prebuilt module on hand but still want to test your project, the schematic below will come in handy.



In the schematic, we have a LM393 op-amp that is a low-power, low offset voltage op-amp that can be powered from a 3.3V or 5V supply. Please note that the analog output voltage of the device will depend on the supply voltage of the device. The main job of this op-amp is to convert the incoming analog signal from the sensor probe to a digital signal. There is also this 10K potentiometer that is used to set a reference voltage for the op-amp, if the input voltage of the sensor goes below the threshold voltage set by the potentiometer, the output of the op-amp goes low. Other than that we have two LEDs. The first one is a power LED and the other one is the trigger LED. The power LED turns on when power is applied to the board and the trigger LED turns on when a certain set threshold is reached. This is how this basic circuit works.

3.2.3.8 ESP32 Code for Rainfall sensor

```
const int mqPin = A0; // Analog pin for sensor

const int DO_Pin=12;

const int buzzerPin = 8; // Digital pin for buzzer

void setup()

{

    pinMode(buzzerPin, OUTPUT);

    pinMode(DO_Pin, INPUT); // Configure D8 pin as a digital input pin

    Serial.begin(9600);

}

void loop()

{
```

```

int sensorValue = analogRead(mqPin);

int threshold= digitalRead(DO_Pin);

Serial.print("threshold_value: ");

Serial.print(threshold);           //prints the threshold_value reached as either LOW or HIGH
Serial.print(", ");

Serial.print("Sensor Value: ");

Serial.println(sensorValue);

delay(100);

// Adjust the threshold value based on your sensor's characteristics

if (threshold==LOW) {

    digitalWrite(buzzerPin, HIGH); // Turn on the buzzer

    delay(200); // Buzzer on time

    digitalWrite(buzzerPin, LOW); // Turn off the buzzer

}

//delay(1000); // Wait before the next reading

}

```

2.3.3 ESP32 Firmware

The ESP32 firmware is the software that runs on the ESP32 microcontroller. It is responsible for controlling the device's hardware components, managing memory, and providing a platform for running user applications.

Types of ESP32 Firmware:

1. Official ESP32 Firmware (provided by Espressif Systems)
2. Arduino Core for ESP32 (based on the Arduino framework)
3. MicroPython Firmware (runs MicroPython interpreter)
4. Lua Firmware (runs Lua interpreter)
5. Custom Firmware (developed by users or third-party vendors)

Key Features:

1. Wi-Fi and Bluetooth connectivity
2. Support for various programming languages (C, C++, MicroPython, Lua, etc.)
3. Low-power modes and sleep functions
4. Secure boot and flash encryption
5. Support for various peripherals (GPIO, SPI, I2C, UART, etc.)

2.3.3.1 Firmware Development:

1. Use the ESP32 SDK (Software Development Kit) for developing custom firmware
2. Utilize the Arduino Core for ESP32 for a more straightforward development process
3. Leverage the MicroPython or Lua firmware for rapid prototyping and development

2.3.3.2 Firmware Updates:

1. OTA (Over-the-Air) updates for remote firmware updates
2. Serial flashing using the ESP32's UART interface
3. USB flashing using the ESP32's USB interface

Developing ESP32 Firmware for IOT Device

2.3.4 ESP32 Firmware modules

- Data collection
- Wifi connection and AWS connection
- Webserver for remote control and OTA update

2.3.4.1 Data collection

Reading Sensor Data every 15 mins

```
String get_sensor_json(void)
{
return    "{\r\n\"Temp\": \" + String(temperature)
          + \" ,\r\n\"Humidity\": \" + String(humidity)
          + \" ,\r\n\"Heat Index\": \" + String(heatIndex)
          + \" ,\r\n\"Dew point\": \" + String(dewPoint)
          + \" ,\r\n\"Feels like\": \" + comfortStatus
          + \" ,\r\n\"Day_Night\": \" + String(day_night)
          + \" ,\r\n\"Moisture_Level\": \" + String(M_per)
          + \" ,\r\n\"R_sensor\": \" + String(R_per)
          + \"\r\n}\";
}

String get_sensor_data(void)
{
return    String(temperature) + \",\" + String(humidity) + \",\" + String(heatIndex)
          + \",\" + String(dewPoint) + \",\" + comfortStatus + \",\" + String(day_night)
          + \",\" + String(M_per) + \",\" + String(R_per);
}

bool getTemperature_Humidity()
{

TempAndHumidity newValues = dht.getTempAndHumidity();
// Check if any reads failed and exit early (to try again).
if (dht.getStatus() != 0)
{
    Serial.println(\"DHT11 error status: \" + String(dht.getStatusString()));
    return false;
}
    heatIndex = dht.computeHeatIndex(newValues.temperature,
newValues.humidity);
    dewPoint = dht.computeDewPoint(newValues.temperature, newValues.humidity);
    float cr = dht.getComfortRatio(cf, newValues.temperature, newValues.humidity);

    switch(cf) {
        case Comfort_OK:
            comfortStatus = \"Comfort_OK\";
            break;
```

```

case Comfort_TooHot:
    comfortStatus = "Comfort_TooHot";
    break;
case Comfort_TooCold:
    comfortStatus = "Comfort_TooCold";
    break;
case Comfort_TooDry:
    comfortStatus = "Comfort_TooDry";
    break;
case Comfort_TooHumid:
    comfortStatus = "Comfort_TooHumid";
    break;
case Comfort_HotAndHumid:
    comfortStatus = "Comfort_HotAndHumid";
    break;
case Comfort_HotAndDry:
    comfortStatus = "Comfort_HotAndDry";
    break;
case Comfort_ColdAndHumid:
    comfortStatus = "Comfort_ColdAndHumid";
    break;
case Comfort_ColdAndDry:
    comfortStatus = "Comfort_ColdAndDry";
    break;
default:
    comfortStatus = "Unknown:";
    break;
};

humidity =newValues.humidity;
temperature=newValues.temperature;
day_night=digitalRead(Day_night_pin);
R_sensorValue = analogRead(Rain_sensor_pin_ADC);
R_per = ((4096-R_sensorValue) / 40.96);

M_sensorValue = analogRead(Moisture_sens_pin);
M_per= ((4096-M_sensorValue) / 40.96);

Serial.println(get_sensor_json());

return true;

```

```
}
```

2.3.4.2 The scheduler

The scheduler is a task that runs at a fixed interval (every 15 minutes) to execute a specific function, `readSensorData()`. This function is responsible for reading data from sensors connected to the ESP32.

Scheduler Details:

- Interval: 15 minutes
- Task Name: Scheduler
- Priority: 1 (low priority)
- Stack Size: 2048 bytes
- Function: `readSensorData()`

Scheduler Functionality:

1. Waits for 15 minutes using `vTaskDelay()`
2. Calls `readSensorData()` to read sensor data
3. Returns to waiting state

This scheduler design allows for:

- Efficient use of system resources
- Non-blocking execution of other tasks
- Consistent and reliable sensor data collection

```
nowSecs = time(nullptr);
struct tm timeinfo;
gmtime_r(&nowSecs, &timeinfo);
if(timeinfo.tm_min!=last_mins)
{
    Serial.print(F("Current time: "));
    Serial.print(asctime(&timeinfo));
    Serial.println("total sec,"+String(nowSecs));

    last_mins=timeinfo.tm_min;
    if(last_mins %15==0)
    {
```

```

last_mins=timeinfo.tm_min;
publishMessage();
}
}

```

2.3.4.3 Wifi connection and AWS connection

The ESP WiFi connection is a functionality that enables the ESP32 microcontroller to connect to a WiFi network. This allows the ESP32 to communicate with other devices and servers over the internet.

Connection Details

- WiFi Mode: Station (STA) mode
- Connection Type: Secure (WPA2)
- Authentication: Password-based
- Encryption: AES

Connection Process

1. Initialize WiFi module
2. Set WiFi mode to Station (STA) mode
3. Connect to WiFi network using SSID and password
4. Obtain IP address and network configuration
5. Verify connection status

WiFi Connection Functions

- `WiFi.begin(ssid, password)`: Connect to WiFi network
- `WiFi.status()`: Get WiFi connection status
- `WiFi.localIP()`: Get local IP address
- `WiFi.SSID()`: Get connected WiFi network SSID

Connection Status:

- `WL_Connected`: Successfully connected to WiFi network
- `WL_Disconnected`: Not connected to WiFi network
- `Connecting`: In process of connecting to WiFi network
- `Failed`: Connection attempt failed


```

void check_connection()
{
    if(WiFi.status() == WL_CONNECTED)
    {
        digitalWrite(led_pin, HIGH);
        return;
    }
    WiFi.begin(ssid, password);
    Serial.println("");
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        digitalWrite(led_pin, HIGH);
        delay(500);
        digitalWrite(led_pin, LOW);
        Serial.print(".");
    }
    setClock();
}

```

Setting up device clock to IST time

```

void setClock()
{
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    Serial.print(F("Waiting for NTP time sync: "));
    time_t nowSecs = time(nullptr);
    while (nowSecs < 8 * 3600 * 2)
    {
        delay(500);
        Serial.print(F("."));
        yield();
        nowSecs = time(nullptr);
    }

    Serial.println();
    nowSecs = time(nullptr);
}

```

```
    struct tm timeinfo;  
    gmtime_r(&nowSecs, &timeinfo);  
    Serial.print(F("Current time: "));  
    Serial.print(asctime(&timeinfo));  
}
```

2.3.4.4 ESP32 IoT Core Connectivity

The ESP32 can connect to AWS IoT Core using the MQTT protocol, enabling secure bi-directional communication between the device and the cloud.

Connection Steps:

1. Setup: Configure AWS IoT Core and create a device certificate, private key, and root CA certificate.
2. Initialize: Initialize the ESP32's WiFi and MQTT client libraries.
3. Connect: Establish a secure MQTT connection to AWS IoT Core using the device certificate, private key, and root CA certificate.
4. Publish: Publish sensor data to an AWS IoT Core topic.
5. Subscribe: Subscribe to an AWS IoT Core topic to receive commands or updates.

2.3.4.5 MQTT Connection Details:

- MQTT Broker: AWS IoT Core endpoint
- MQTT Port: 8883 (secure)
- MQTT Protocol: MQTT v3.1.1
- Client ID: ESP32 device ID
- Username: AWS IoT Core device username
- Password: AWS IoT Core device password

AWS IoT Core Benefits:

- Secure: End-to-end encryption and secure authentication
- Scalable: Handle large numbers of devices and messages
- Reliable: Guaranteed message delivery and retries
- Flexible: Support for various protocols and device types

By connecting to AWS IoT Core, the ESP32 can:

- Send sensor data to the cloud for processing and analysis
- Receive commands and updates from the cloud
- Integrate with other AWS services and applications
- Enable remote monitoring and control of devices
- Webserver for remote control and OTA update

ESP32 AWS connection code

Void **aws_connection()**

```
{
    if(client.connected())
    {
        digitalWrite(led_pin, HIGH);
        return;
    }

    net.setCACert(AWS_CERT_CA);
    net.setCertificate(AWS_CERT_CRT);
    net.setPrivateKey(AWS_CERT_PRIVATE);
    client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
    client.begin(AWS_IOT_ENDPOINT, 8883, net);
}
```

```

    client.onMessage(messageHandler);
Serial.print("Connecting to AWS IOT");
while (!client.connect(THINGNAME))
{
    Serial.print(".");
    delay(100);
    digitalWrite(led_pin, HIGH);
    delay(100);
    digitalWrite(led_pin, LOW);

}
if(!client.connected())
{
    Serial.println("AWS IoT Timeout!");
}
Serial.println("AWS IoT Connected!");
}

```

2.3.4.6 Webserver for remote control and OTA update

ESP32 Webserver:

- Runs a web server on the ESP32
- Serves HTML, CSS, and JavaScript files

It Allows users to:

- View sensor data in real-time
- Update firmware Over-The-Air (OTA)
- Control device settings

Uses:

- ESPAsyncWebServer library

- ArduinoOTA library for OTA updates

Benefits:

- Remote monitoring and control
- Easy firmware updates
- Flexible user interface

2.3.4.7 ESP32 WebServer Code

```

/*return index page which is stored in serverIndex */
server.on("/", HTTP_GET, []()
{
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
});
server.on
("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
});

server.on("/sensordata", HTTP_GET, sensordataindex);
server.on("/motorcontrol", HTTP_GET, GPIO_control);
server.on("/M1on", HTTP_GET, M1_on);
server.on("/M2on", HTTP_GET, M2_on);
server.on("/M1off", HTTP_GET, M1_off);
server.on("/M2off", HTTP_GET, M2_off);

/*handling uploading firmware file */
server.on("/update", HTTP_POST, []()
{
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
}, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
        Serial.printf("Update: %s\n", upload.filename.c_str());
        if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available
size
            Update.printError(Serial);
        }
    } else if (upload.status == UPLOAD_FILE_WRITE) {
        /* flashing firmware to ESP*/

```

```

        if (Update.write(upload.buf, upload.currentSize) !=
upload.currentSize) {
            Update.printError(Serial);
        }
    } else if (upload.status == UPLOAD_FILE_END) {
        if (Update.end(true)) { //true to set the size to the current
progress
            Serial.printf("Update Success: %u\nRebooting...\n",
upload.totalSize);
        } else {
            Update.printError(Serial);
        }
    }
});
server.begin();
}

```

2.3.4.8 Login Page for ESP-32 webserver

The ESP32 login page is a web-based interface that allows users to authenticate and access the ESP32's web server. Here's a description of the login page:

Title: ESP32 Login

Fields:

1. Username: A text input field for the user to enter their username.
2. Password: A password input field for the user to enter their password.

Buttons:

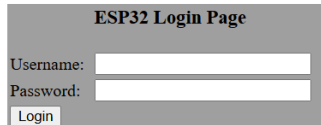
Login: A submit button to authenticate the user.

Functionality:

- When the user enters their username and password, the login form sends a POST request to the ESP32's web server.
- The ESP32 verifies the username and password.

- If the credentials are correct, the ESP32 grants access to the web server and displays a success message.
- If the credentials are incorrect, the ESP32 displays an error message.

192.168.29.122
LMS - University Of...

A screenshot of a web browser displaying the 'ESP32 Login Page'. The page has a light gray background. At the top, the title 'ESP32 Login Page' is centered. Below the title, there are two input fields: 'Username:' and 'Password:'. Each field has a white text box with a gray border. Below the 'Password:' field, there is a 'Login' button with a gray background and white text.

Code

```
Const char* loginIndex =
```

```
"<form name='loginForm'>"
```

```
"<table width='20%' bgcolor='A09F9F' align='center'>"
```

```
"<tr>"
```

```
"<td colspan=2>"
```

```
"<center><font size=4><b>ESP32 Login Page</b></font></center>"
```

```
"<br>"
```

```
"</td>"
```

```
"<br>"
```

```
"<br>"
```

```
"</tr>"
```

```
"<tr>"
```

```
"<td>Username:</td>"
```

```
"<td><input type='text' size=25 name='userid'><br></td>"
```

```
"</tr>"
```

```
"<br>"
```

```
"<br>"
```

```
"<tr>"
```

```

" <td>Password:</td>"

" <td><input type='Password' size=25 name='pwd'><br></td>"

" <br>"

" <br>"

"</tr>"

"<tr>"

" <td><input type='submit' onclick='check(this.form)' value='Login'></td>"

"</tr>"

"</table>"

"</form>"

"<script>"

"function check(form)"

"{"

"if(form.userid.value=='admin' && form.pwd.value=='admin')"

"{"

"window.open('/serverIndex')"

"}"

"else"

"{"

" alert('Error Password or Username')/*displays error message*/"

"}"

"}"

"</script>";

```

2.3.4.9 View sensor data in real-time from ESP32 webserver

The ESP32 web server streams sensor data to a web page in real-time, allowing users to monitor sensor readings instantly.

Features:

Real-time updates: Sensor data is updated on the web page in real-time, without requiring manual refresh.

Automatic data transmission: The ESP32 web server automatically sends sensor data to the client at regular intervals.

Web-based interface: Sensor data is displayed on a web page, accessible from any device with a web browser.

Benefits:

Instant monitoring: Users can monitor sensor data in real-time, allowing for prompt responses to changes.

Remote access: Users can access sensor data from anywhere, using any device with a web browser.

Easy integration: The ESP32 web server can be integrated with other systems and services, such as IoT platforms and databases.

Technical Details:

Communication protocol: WebSockets used for real-time communication between the client and server.

Data format: Sensor data is transmitted in plain text or JSON format.

```
{
  "Temp":29.80,
  "Humidity":94.00,
  "Heat Index":41.38,
  "Dew point":28.74,
  "Feels like":Comfort_HotAndHumid,
  "Day_Night":1,
  "Moisture_Level":0.02,
  "R_sensor":0.02
}
```

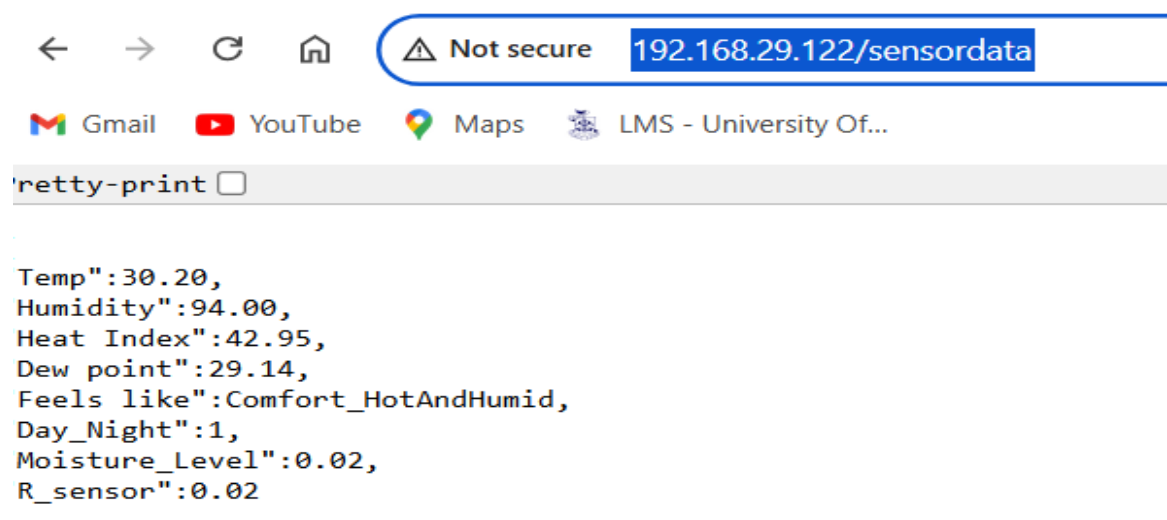
Update interval: The interval at which sensor data is sent to the client can be adjusted.

Applications:

Industrial monitoring: Real-time sensor data can be used to monitor industrial processes and equipment.

Environmental monitoring: Sensor data can be used to monitor environmental conditions, such as temperature and humidity.

Home automation: Real-time sensor data can be used to control and monitor home automation systems.



ESP-32 Code

```
void sensordataindex()
{
    server.send(200, "text/json", get_sensor_json());
}

String get_sensor_json(void)
{
    return "{\r\n\"Temp\": \" + String(temperature)
           + \" ,\r\n\"Humidity\": \" + String(humidity)
           + \" ,\r\n\"Heat Index\": \" + String(heatIndex)
           + \" ,\r\n\"Dew point\": \" + String(dewPoint)
           + \" ,\r\n\"Feels like\": \" + comfortStatus
           + \" ,\r\n\"Day_Night\": \" + String(day_night)
           + \" ,\r\n\"Moisture_Level\": \" + String(M_per)
           + \" ,\r\n\"R_sensor\": \" + String(R_per)
```

```
        + "\r\n}";  
    }
```

2.3.4.10 Control and Monitor Equipements in real-time from ESP32 webserver

The ESP32 web server allows users to control and monitor equipment in real-time, remotely, using a web-based interface.

Features:

Real-time monitoring: Equipment status and sensor data are displayed in real-time on the web page.

Remote control: Users can control equipment remotely, using buttons or sliders on the web page.

Technical Details:

Communication protocol: WebSockets used for real-time communication between the client and server.

Control protocol: Equipment is controlled using digital or analog outputs from the ESP32.

Applications:

Industrial automation: Control and monitor industrial equipment, such as motors, pumps, and valves.

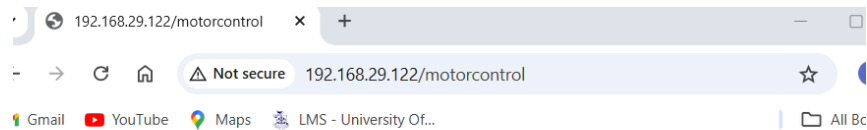
Home automation: Control and monitor home appliances, such as lights, thermostats, and security systems.

Agricultural monitoring: Monitor and control agricultural equipment, such as irrigation systems and greenhouses.

Equipment Examples:

1. Motors
2. Pumps
3. Valves
4. Greenhouses

ESP32 Control Page

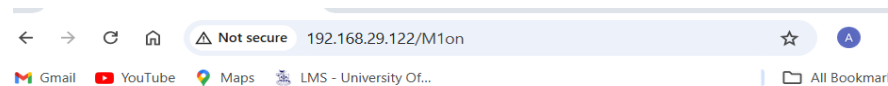


Samrt Farming remote control Server

Motor 1 - State off



Motor 2 - State off



Samrt Farming remote control Server

Motor 1 - State on



Motor 2 - State off



ESP32 Control page code

```
void GPIO_control()
{
    publishMessage();
    String send_str="<!DOCTYPE html><html>"
        "<head><meta name=\"viewport\" content=\"width=device-  
width, initial-scale=1\">"
        "<link rel=\"icon\" href=\"data:;\">"
        "<style>html { font-family: Helvetica; display: inline-  
block; margin: 0px auto; text-align: center;}"
        ".button { background-color: #4CAF50; border: none;  
color: white; padding: 16px 40px;"
        "text-decoration: none; font-size: 30px; margin: 2px;  
cursor: pointer;}"
        ".button2 {background-color: #555555;}</style></head>"
```

```

"<body><h1>Smart Farming remote control Server</h1>"
"<p>Motor 1 - State " + M1State + "</p>";

    if (M1State=="off")
    {
        send_str=send_str+"<p><a href=\"/M1on\"><button
class=\"button\">ON</button></a></p>";
    }
    else
    {
        send_str=send_str+"<p><a href=\"/M1off\"><button
class=\"button button2\">OFF</button></a></p>";
    }
    send_str=send_str+"<p>Motor 2 - State " + M2State +
"</p>";

    if (M2State=="off")
    {
        send_str=send_str+"<p><a href=\"/M2on\"><button
class=\"button\">ON</button></a></p>";
    }
    else
    {
        send_str=send_str+"<p><a href=\"/M2off\"><button
class=\"button button2\">OFF</button></a></p>";
    }
    send_str=send_str+"</body></html>";
    server.send(200, "text/html", send_str);
}

```

2.4 AWS cloud services

AWS IoT Core: A cloud-based platform that receives data from the ESP32, processes it, and triggers actions based on predefined rules.

AWS IoT Core is a managed cloud service that enables connected devices to easily and securely interact with cloud applications and other devices.

Key Features

1. Device Management: Register, organize, and remotely manage IoT devices.
2. Data Processing: Process and transform data from devices in real-time.
3. Analytics: Integrate with AWS analytics services for data insights.
4. Security: End-to-end encryption, mutual authentication, and authorization.
5. Scalability: Handle large volumes of devices and data.

Benefits

1. Easy Integration: Connect devices to the cloud with minimal effort.
2. Real-time Insights: Process and analyze data in real-time.
3. Secure: Robust security features protect data and devices.
4. Scalable: Handle growing numbers of devices and data.

Technical Details

1. Device Connectivity: Supports MQTT, HTTP, and WebSockets protocols.
2. Device Shadow: Virtual representation of device state.
3. Thing Registry: Managed registry for device metadata and attributes.
4. Rules Engine: Managed service for processing and transforming data.
5. Device Defender: Managed service for detecting and responding to security threats.

Pricing

1. Pay-per-Use: Pay only for the data and messages used.
2. Tiered Pricing: Discounts for large volumes of data and messages.

Use Cases

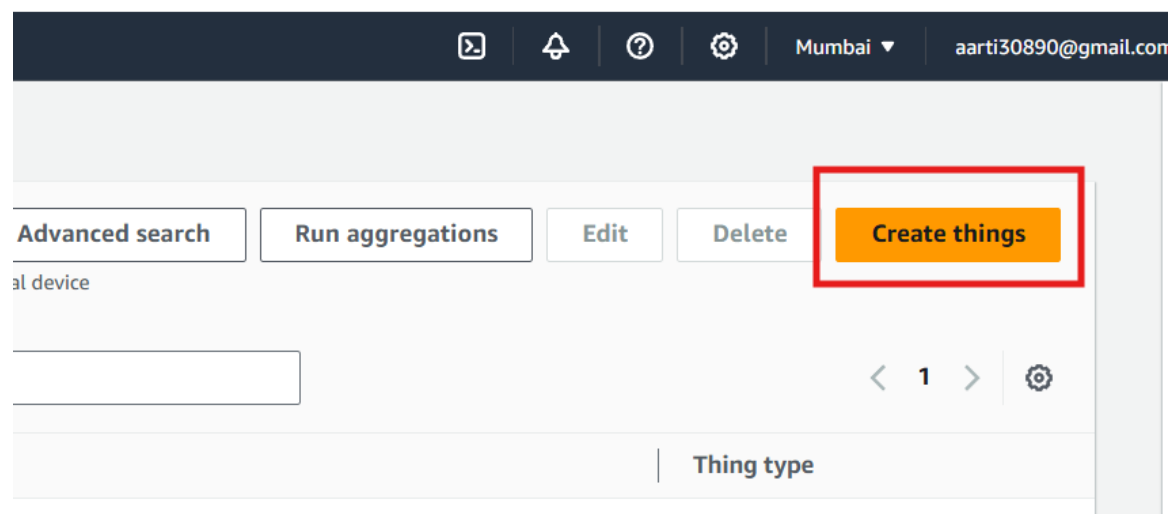
1. Industrial Automation: Monitor and control industrial equipment.
2. Smart Homes: Control and monitor home appliances and devices.
3. Wearables: Collect and analyze data from wearable devices.
4. Transportation: Monitor and manage vehicles and fleets.
5. Smart Farming: Monitor and optimize agricultural operations.

2.4.1 Steps to create an IoT Core thing in AWS

Navigate to IoT Core: Click on the "Services" dropdown menu and select "IoT Core" under the "Internet of Things" section.

Click on "Manage" and then "Things": In the IoT Core dashboard, click on the "Manage" dropdown menu and select "Things".

Click on "Create thing": Click on the "Create thing" button to start the process of creating a new thing.

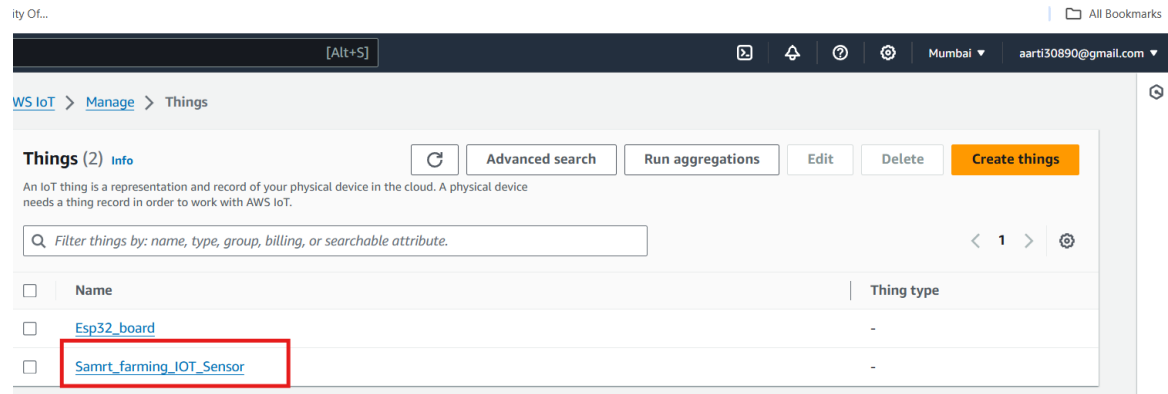


Enter thing name and description: Enter a name and description for your thing.

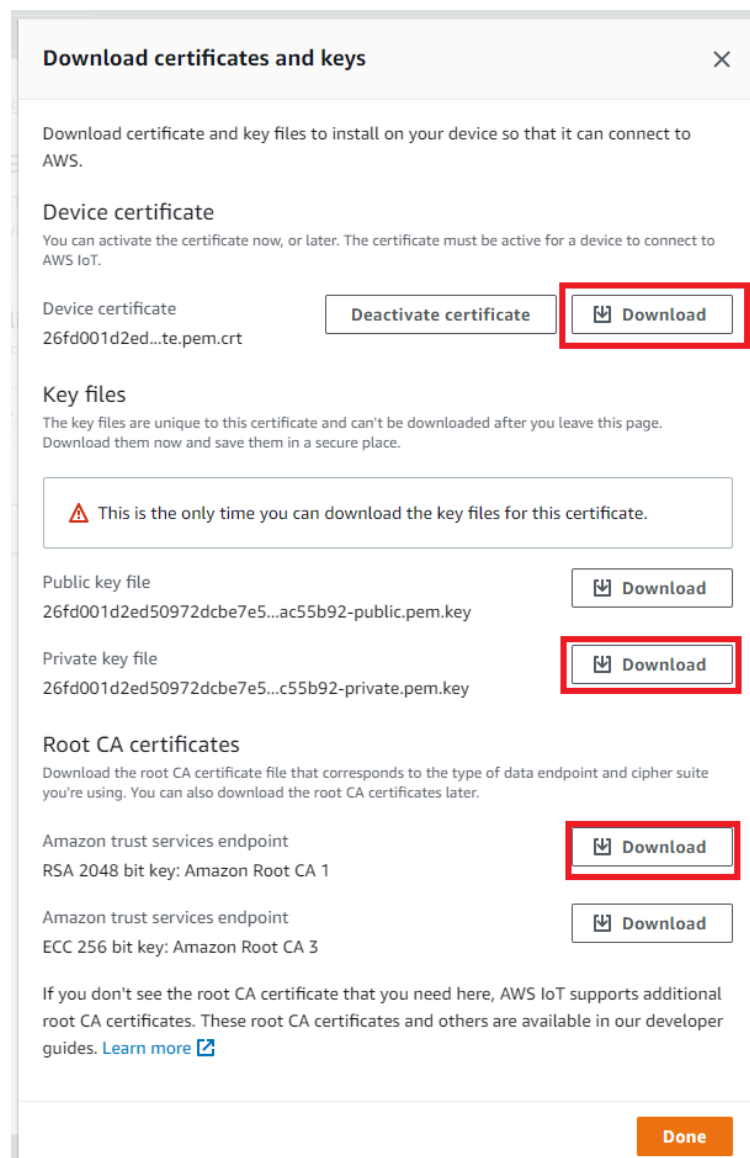
Choose a thing type: Select a thing type from the dropdown menu or create a new one.

Configure thing attributes: Configure any additional attributes for your thing, such as location or serial number.

Create thing: Click on the "Create thing" button to create the thing.



Create certificate and keys: Create a certificate and keys for your thing to use for authentication.



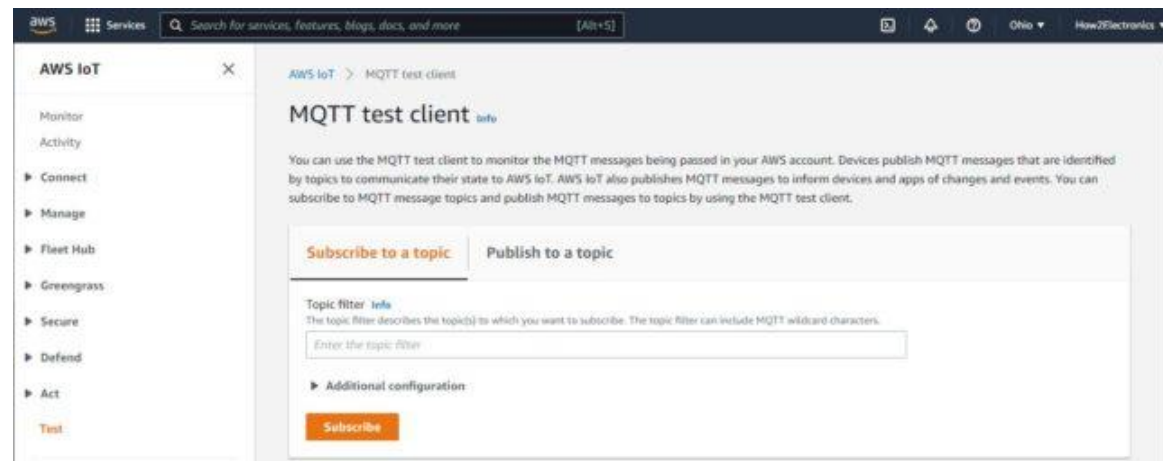
Activate thing: Activate your thing to start sending and receiving data.

Attach policies: Attach policies to your thing to define its permissions and behaviors.

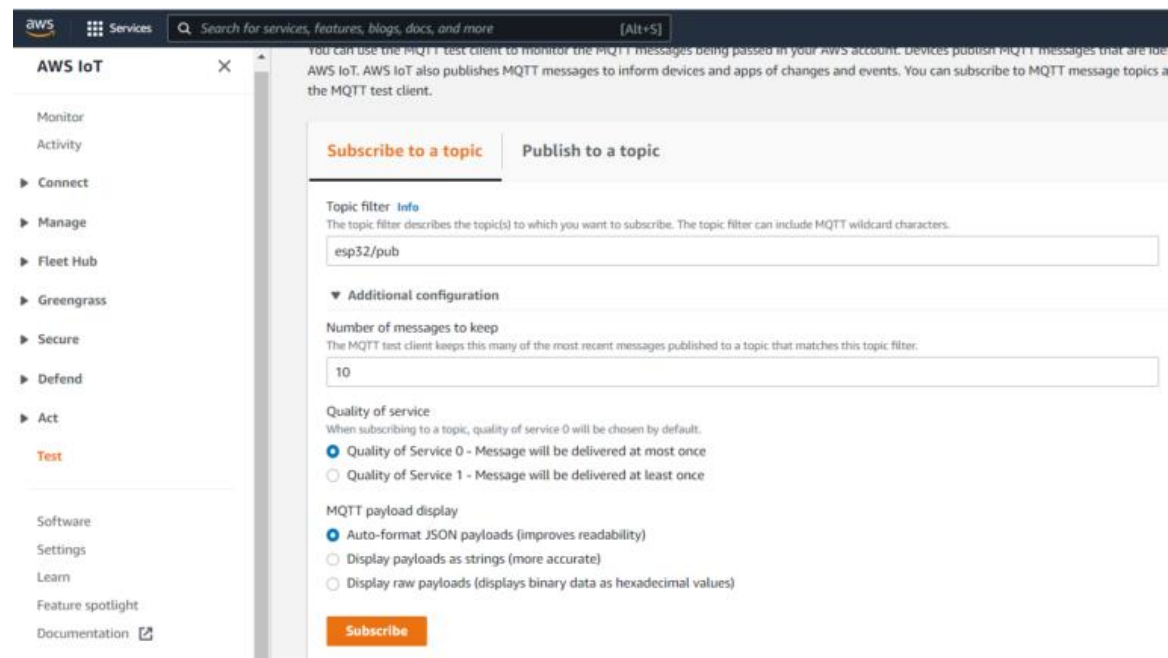
2.4.2 Testing AWS IOT core device connection

Subscribing Sensor Data to AWS Dshboard

The same thing should also be posted to the AWS Server. To check that, go to the test section of AWS Dashboard. Under the test section, we have an option for subscribe and publish.

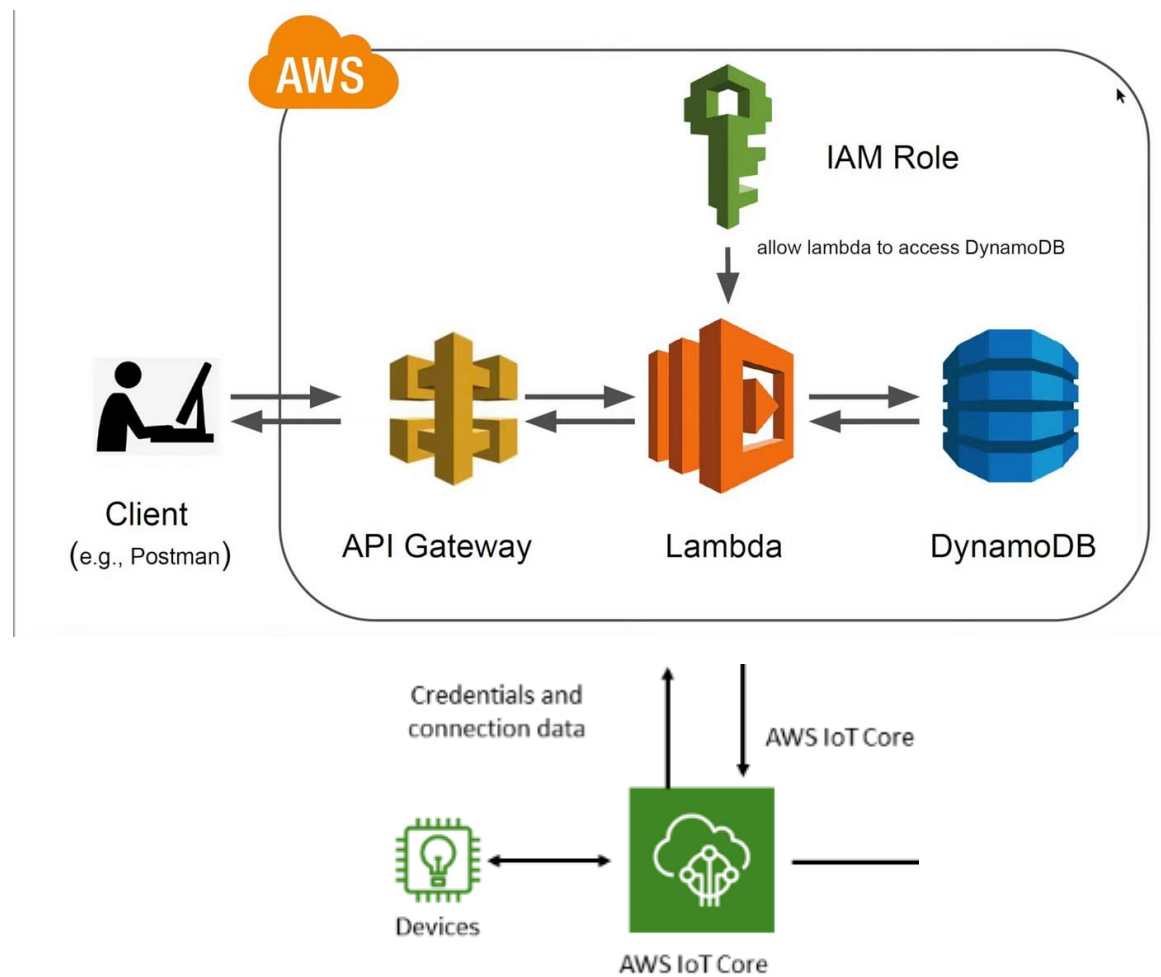


Now to see the data, you need to subscribe to a topic. For that type “esp32/pub“ under the topic filter section. In the additional configuration, you can make changes if you want.



Then click on subscribe. When you hit the subscribe button, immediately the data from ESP32 will be uploaded to AWS Dashboard. Thus, you have successfully sent the Sensor data to Amazon AWS IoT Core using ESP32.

2.4.3 Storing and Retrieving Data from Sensor to DynamoDB



2.4.3.1 AWS Lambda

AWS Lambda is a serverless compute service that runs code in response to events, such as changes to data in an Amazon S3 bucket or an Amazon DynamoDB table.

Key Features:

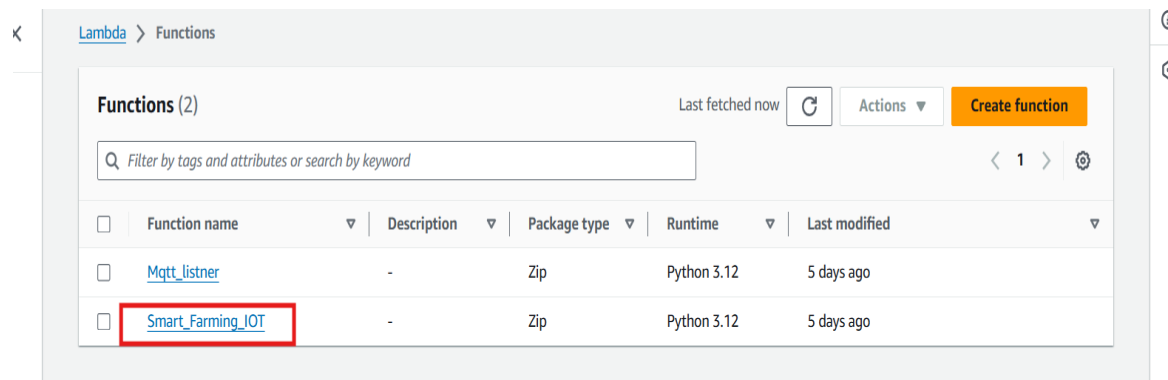
- Serverless: No servers to manage or provision
- Event-driven: Runs code in response to events
- Scalable: Automatically scales to handle large workloads
- Cost-effective: Only pay for the compute time consume

Use Cases:

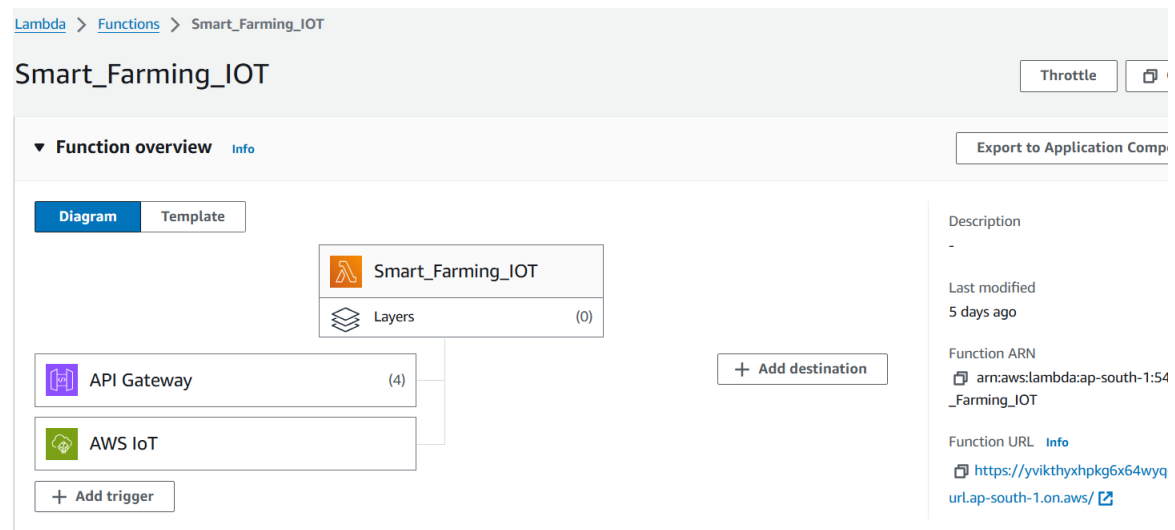
- Real-time data processing
- API backend
- Scheduled tasks
- IoT data processing
- Machine learning

Benefits:

- Increased productivity
- Reduced costs
- Improved scalability
- Enhanced reliability



2.4.3.2 AWS Lambda Trigger with AWS API gateway Setups



Below are steps how to trigger an AWS Lambda function using a REST API:

Step 1: Create an API Gateway

- Go to the AWS Management Console and navigate to API Gateway
- Click "Create API" and choose "REST API"
- Define the API endpoint and method (e.g. POST /trigger-lambda)

Step 2: Integrate with Lambda

- In the API Gateway console, click "Integration Request"
- Choose "Lambda Function" as the integration type
- Select the Lambda function you want to trigger

Step 3: Deploy the API

- Click "Deploy API" and choose a stage (e.g. "test")
- Note the API endpoint URL (e.g. (link unavailable))

Step 4: Test the API

- Use a tool like Postman or cURL to send a POST request to the API endpoint URL
- The Lambda function should be triggered and execute

2.4.4.2 AWS Lambada Code

```
import json
import logging
import boto3

logger = logging.getLogger()
logger.setLevel(logging.INFO)

dynamodb_client = boto3.client('dynamodb')

def lambda_handler(event, context):

    logger.info("##### Lambda API event ##### - %s" % event)
    logger.info("##### Lambda API context ##### - %s" % context)

    Sensordata=event['Sensordata']

    if Sensordata== "IOT_sensor": payload_data="{\"data\": {\"S\": \"payload_data\"},
    \"Date_time\": {\"N\": \"12344567890\"}}}"

    payload = str(event)

    Date_time = event['time_stamp']

    payload_data = payload_data.replace("payload_data", payload)
    payload_data = payload_data.replace("12344567890",Date_time)

    http_body=json.loads(payload_data)

    #http_body['data']=payload

    #http_body['Date_time']=Date_time

    data = dynamodb_client.put_item(TableName='Sensor_data',Item=http_body)

    # Get the Http Method from API Request
```

```

if Sensordata == "Search":

    logger.info('Handling Customer API - Get request')

    Start = event['Start']

    End = event['End']

    response = dynamodb_client.scan(TableName='Sensor_data')

    response_body=""

    for item in response['Items']:

        response_body=response_body+ json.dumps(item.get('data'))+", "


    logger.info("#### Lambda API table data #### - %s" % response_body)


    return {

        'statusCode': 200,

        'body': json.dumps(response_body)

    }


return {

    'statusCode': 200,

    'body': json.dumps('Success')

}

```

2.4.4 Store / retrieve records in DynamoDB

DynamoDB is a fast, fully managed NoSQL database service provided by Amazon Web Services (AWS).

Key Features:

- **High Performance:** DynamoDB provides fast and consistent performance, with single-digit millisecond latency.
- **Fully Managed:** DynamoDB is a fully managed service, meaning AWS handles provisioning, patching, and maintenance.
- **NoSQL:** DynamoDB is a NoSQL database, allowing for flexible schema design and easy data storage.
- **Scalability:** DynamoDB automatically scales to handle large amounts of data and traffic.
- **High Availability:** DynamoDB provides high availability and durability, with automatic replication and backup.

Use Cases:

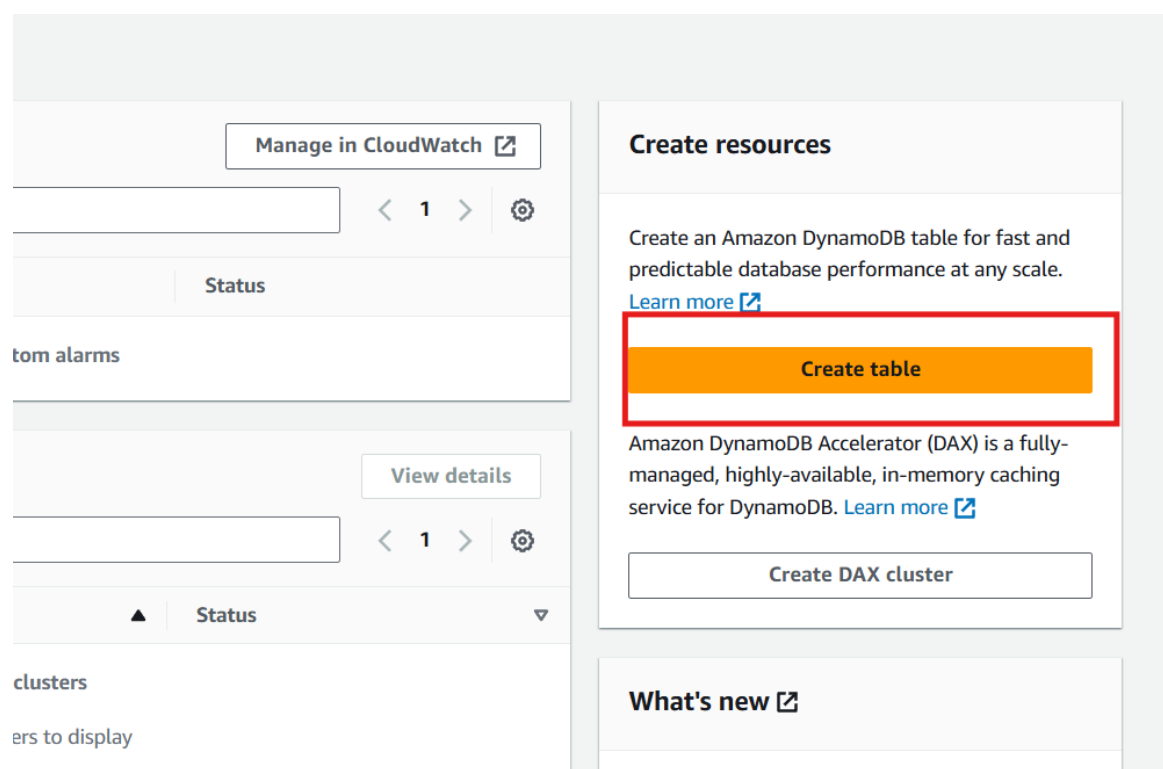
- **Real-time Applications:** DynamoDB is well-suited for real-time applications, such as gaming, live updates, and streaming data.
- **Mobile and Web Applications:** DynamoDB is a good choice for mobile and web applications, providing fast and consistent performance.
- **IoT Data Storage:** DynamoDB can handle large amounts of IoT data, providing fast and scalable storage.
- **Big Data Analytics:** DynamoDB can be used as a data store for big data analytics, providing fast and scalable storage.

Benefits:

- **Fast Performance:** DynamoDB provides fast and consistent performance, allowing for real-time applications.

- **Scalability:** DynamoDB automatically scales to handle large amounts of data and traffic.
- **High Availability:** DynamoDB provides high availability and durability, with automatic replication and backup.
- **Cost-Effective:** DynamoDB is a cost-effective solution, with pay-per-use pricing and no upfront costs.

2.4.1 Create Table in DynamoDB



The screenshot shows the AWS Management Console interface for DynamoDB. On the left, there are sections for 'Manage in CloudWatch', 'Status', and 'View details'. On the right, the 'Create resources' section is visible, featuring a prominent orange 'Create table' button highlighted with a red rectangle. Below it, there is a link to 'Learn more' and a section for 'Amazon DynamoDB Accelerator (DAX)' with a 'Create DAX cluster' button. At the bottom, there is a 'What's new' section.

Create resources

Create an Amazon DynamoDB table for fast and predictable database performance at any scale. [Learn more](#)

Create table

Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. [Learn more](#)

Create DAX cluster




What's new

DynamoDB > Tables

Tables (3) [Info](#)

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write cap
<input type="checkbox"/>	IOT_farming_data	Active	Data (S)	-	0	Off	Provisioned (1)	Provisioned
<input type="checkbox"/>	Sensor_data	Active	Date_time (N)	-	0	Off	Provisioned (1)	Provisioned
<input type="checkbox"/>	Sensor_data_15mins	Active	id (S)	-	0	Off	Provisioned (1)	Provisioned

2.4.1.1 Sensor Data from DynamoDB tables

Items returned (50)			Actions ▼	Create item
		< 1 ... >  		
<input type="checkbox"/>	Date_time (Number) ▲	data ▼		
<input type="checkbox"/>	1724927400	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1724927400', 'Temp': '28.50', 'Humidity': '96.00', 'Heat_Index': '37.15', 'Dew_po...		
<input type="checkbox"/>	1724936400	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1724936400', 'Temp': '28.00', 'Humidity': '96.00', 'Heat_Index': '35.40', 'Dew_po...		
<input type="checkbox"/>	1724940900	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1724940900', 'Temp': '28.00', 'Humidity': '95.00', 'Heat_Index': '35.16', 'Dew_po...		
<input type="checkbox"/>	1724949900	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1724949900', 'Temp': '27.60', 'Humidity': '96.00', 'Heat_Index': '34.07', 'Dew_po...		
<input type="checkbox"/>	1724995800	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1724995800', 'Temp': '28.00', 'Humidity': '93.00', 'Heat_Index': '34.69', 'Dew_po...		
<input type="checkbox"/>	1724996700	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1724996700', 'Temp': '28.00', 'Humidity': '92.00', 'Heat_Index': '34.46', 'Dew_po...		
<input type="checkbox"/>	1725000300	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1725000300', 'Temp': '28.50', 'Humidity': '92.00', 'Heat_Index': '36.09', 'Dew_po...		
<input type="checkbox"/>	1725034500	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1725034500', 'Temp': '29.80', 'Humidity': '92.00', 'Heat_Index': '40.70', 'Dew_po...		
<input type="checkbox"/>	1725039000	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1725039000', 'Temp': '29.80', 'Humidity': '93.00', 'Heat_Index': '41.04', 'Dew_po...		
<input type="checkbox"/>	1725047100	{ 'Sensordata': 'IOT_sensor', 'time_stamp': '1725047100', 'Temp': '29.30', 'Humidity': '93.00', 'Heat_Index': '39.17', 'Dew_po...		

Chapter III

3.1 Findings

3.1.1 statistical analysis in smart farming using IoT:

Smart Farming IoT Data

- Sensors collect data on temperature, humidity, soil moisture, crop health, and weather
- IoT devices transmit data to a central platform for analysis
- Statistical analysis is applied to the data to gain insights and make informed decisions

3.1.2 Statistical Analysis Techniques

- Descriptive statistics: summarize and describe the data (mean, median, mode, standard deviation)
- Inferential statistics: make predictions and inferences about the population (hypothesis testing, confidence intervals)
- Regression analysis: model the relationship between variables (predict crop yield based on weather and soil conditions)
- Time series analysis: analyze data over time (predict future crop yields based on historical data)
- Machine learning algorithms: classify, cluster, and predict outcomes (predict crop diseases, detect anomalies)

3.1.3 Popular graph UI libraries for statistical analysis

D3.js: A popular JavaScript library for producing dynamic, interactive data visualizations.

Plotly: A high-level, declarative charting library for Python, R, and JavaScript.

Matplotlib: A popular Python library for creating static, animated, and interactive visualizations.

Seaborn: A Python visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.

Bokeh: A Python library for creating interactive, web-based visualizations.

Altair: A Python library for creating interactive, web-based visualizations, with a simple and concise API.

ECharts: A popular JavaScript library for creating interactive, web-based visualizations.

Highcharts: A popular JavaScript library for creating interactive, web-based visualizations.

These libraries can be used to create a variety of statistical graphs, including:

- Scatter plots
- Line charts
- Bar charts
- Histograms
- Heatmaps
- Box plots

- Violin plots
- Scatterplot matrices

They also provide features like:

- Interactive zooming and panning
- Hover-over text and tooltips
- Animations and transitions
- Customizable colors, fonts, and layouts
- Support for large datasets

These libraries can be used in various applications, including:

- Data science and machine learning
- Business intelligence and analytics
- Scientific research and publication
- Education and teaching
- Web development and design

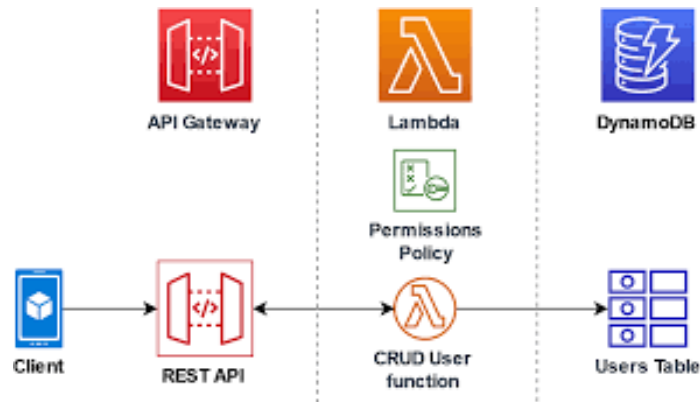
We are going to use Plotly to create data graphs from collected data stored into dynamoDB

3.2. Plotly

Plotly is a popular open-source data visualization library used for creating interactive, web-based visualizations. It supports over 40 unique chart types, including line plots, scatter plots, bar charts, histograms, heatmaps, box plots, violin plots, and many more.

Below is architecture of Data analysis module

3.2.1 Architecture of Data analysis module



Using Python SDK for AWS DynamoDB tables are saved to local device as CSV file

- Use the boto3 library to connect to DynamoDB
- Use the scan method to retrieve the data
- Use the csv library to write the data to a CSV file

3.2.2 Code to store Table data to local Device

```

dynamodb_client =
boto3.client('dynamodb',aws_access_key_id="AKIAX7MBR6XGUUWWDKXF"
,aws_secret_access_key="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",r
egion_name="ap-south-1")
response = dynamodb_client.scan(TableName='Sensor_data')
response_body = "
data_file = open('data_file.csv', 'w')

# create the csv writer object
csv_writer = csv.writer(data_file)

# Counter variable used for writing
# headers to the CSV file
count = 0

for Sensor_data_record in response['Items']:
    Sensor_data= json.loads(Sensor_data_record['data']['S'].replace("'", "\"").strip())
    if count == 0:
        # Writing headers of CSV file
        header = Sensor_data.keys()
        csv_writer.writerow(header)
        count += 1

```

```

# Writing data of CSV file
if Sensor_data['time_stamp']!= '0':

    Sensor_data['time_stamp'] =
datetime.datetime.utcfromtimestamp(int(Sensor_data['time_stamp']) +
19800).strftime('%Y-%m-%dT%H:%M:%S')

    csv_writer.writerow(Sensor_data.values())

data_file.close()
with open(
    "data_file.csv", 'r') as r, open(
'data_file_filtered.csv', 'w') as o:
    for line in r:
        # strip() function
        if "\"time_stamp\":"\"0\"" not in line:
            if line.strip():
                o.write(line)

df = pd.read_csv("data_file_filtered.csv")
sorted_df = df.sort_values(by=["time_stamp"], ascending=True)
sorted_df.to_csv('Data_sorted.csv', index=False)
df = pd.read_csv('Data_sorted.csv')
df.drop(0);

```

Sensordata	time_stamp	Temp	Humidity	Heat_Index	Dew_point	Feels_like
IOT_sensor	2024-08-28T13:33:21	28.9	93.0	37.73	27.67	Comfort_HotAnd
IOT_sensor	2024-08-28T16:49:41	28.5	94.0	36.61	27.45	Comfort_HotAnd
IOT_sensor	2024-08-28T16:49:42	28.5	94.0	36.61	27.45	Comfort_HotAnd
IOT_sensor	2024-08-28T16:51:29	28.5	94.0	36.61	27.45	Comfort_HotAnd
IOT_sensor	2024-08-28T16:51:30	28.5	94.0	36.61	27.45	Comfort_HotAnd
IOT_sensor	2024-08-28T17:00:00	28.0	95.0	35.16	27.14	Comfort_HotAnd
IOT_sensor	2024-08-28T17:15:02	28.0	95.0	35.16	27.14	Comfort_HotAnd
IOT_sensor	2024-08-28T17:18:02	28.5	95.0	36.88	27.63	Comfort_HotAnd
IOT_sensor	2024-08-28T17:30:00	27.6	96.0	34.07	26.92	Comfort_HotAnd
IOT_sensor	2024-08-28T17:45:00	27.6	96.0	34.07	26.92	Comfort_HotAnd

3.2.2.1 CSV DataTable from DynamoDB

3.2.3 Plotting Charts from Data tables

Plotly supports various formats for data input and output. Here are some examples:

Input formats:

1. Pandas DataFrames: Plotly supports direct input from Pandas DataFrames, making it easy to integrate with popular data analysis libraries.
2. NumPy arrays: Plotly can accept NumPy arrays as input for chart data.
3. CSV files: Plotly can read data directly from CSV files.
4. Excel files: Plotly can read data from Excel files (.xlsx, .xls).
5. JSON data: Plotly can accept JSON data as input.

Output formats:

1. HTML: Plotly charts can be output as HTML files, making it easy to embed them in web pages.
2. PNG images: Plotly charts can be saved as PNG images.
3. SVG images: Plotly charts can be saved as SVG images.
4. PDF documents: Plotly charts can be saved as PDF documents.
5. Interactive web pages: Plotly charts can be output as interactive web pages using Plotly's built-in web server.

3.2.3.1 Types of Charts supported by Plotly

Line Chart

- Used for: Trend analysis, time series data
- Data requirements: Date or category on x-axis, numeric values on y-axis
- Features: Connects data points with lines, shows trends and patterns

Bar Chart

- Used for: Categorical data, comparisons
- Data requirements: Categories on x-axis, numeric values on y-axis
- Features: Displays bars for each category, compares values

Scatter Plot

- Used for: Relationships, correlations
- Data requirements: Numeric values on both x and y axes
- Features: Displays data points as dots, reveals relationships and correlations

Histogram

- Used for: Distributions, frequency
- Data requirements: Numeric values
- Features: Displays bars for each range, shows distribution and frequency

Heatmap

- Used for: Matrix data, patterns
- Data requirements: Numeric values for each cell
- Features: Displays color-coded matrix, reveals patterns and correlations

Pie Chart

- Used for: Proportional data, percentages
- Data requirements: Categories and corresponding percentages
- Features: Displays pie slices for each category, shows proportions

Stacked Chart

- Used for: Cumulative data, contributions
- Data requirements: Categories and numeric values
- Features: Displays stacked bars or lines, shows cumulative values

Scatter Plot from IOT sensor Data using Plotly

3.2.4 Creating a scatter plot from IoT sensor data using Plotly:

Step 1: Import Libraries

- Import Plotly Express (px) for creating the scatter plot
- Import Pandas (pd) for data manipulation and analysis

Step 2: Prepare Data

- Collect IoT sensor data (e.g., temperature, humidity, pressure) from your device or database
- Organize data into a Pandas DataFrame with columns for time, temperature, humidity, etc.

Step 3: Create Scatter Plot

- Use `px.scatter()` to create a scatter plot with time on the x-axis and temperature on the y-axis
- Customize plot title, labels, and appearance as needed

Step 4: Customize Plot

- Use various options to customize the plot, such as:
- `color`: change marker color
- `symbol`: change marker symbol
- `size`: change marker size
- `hover_name`: add hover text
- `hover_data`: add additional hover data

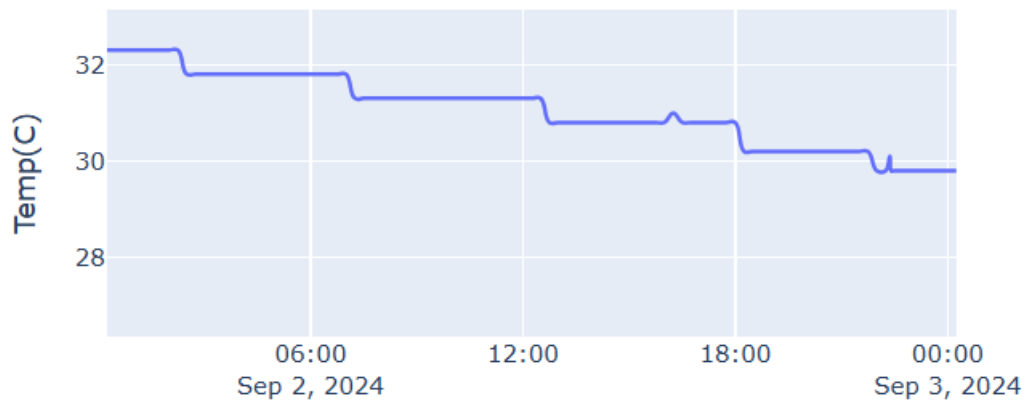
Step 5: Display Plot

- Use **`fig.show()`** to display the scatter plot

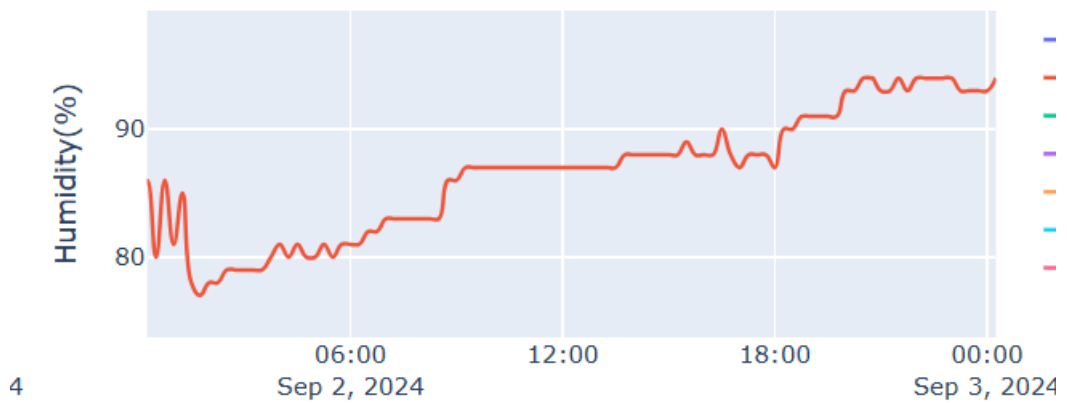
3.2.5 Dashboard from Data collected from IOT sensor



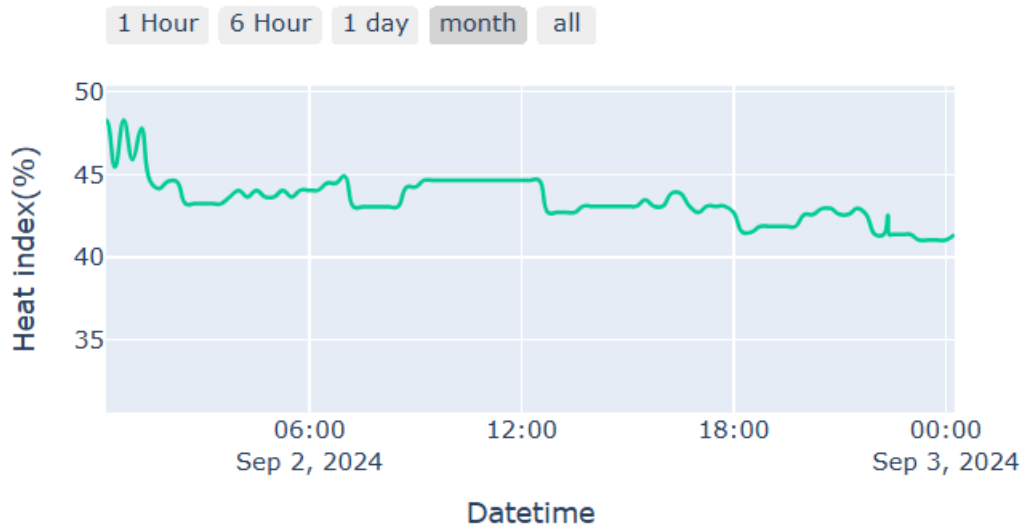
3.2.5.1 Dashboard



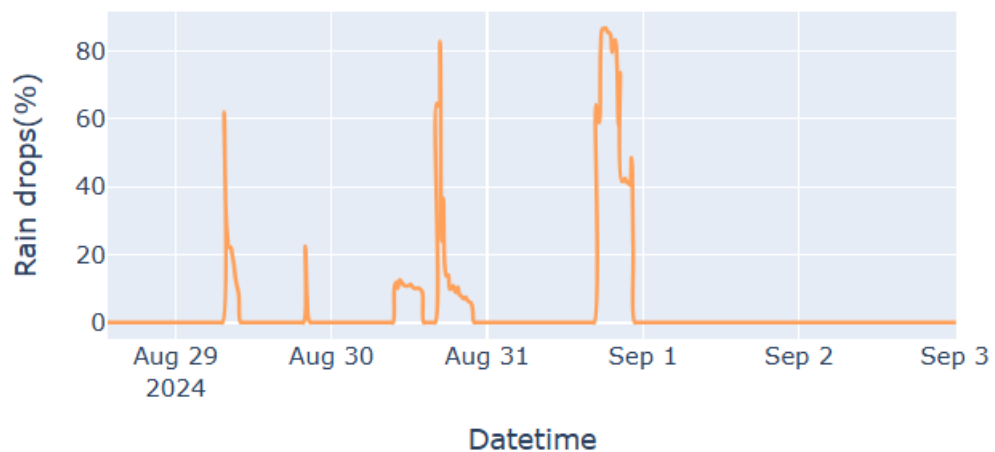
Temperature graph



Humidity Graph



Heat index



Rain Fall

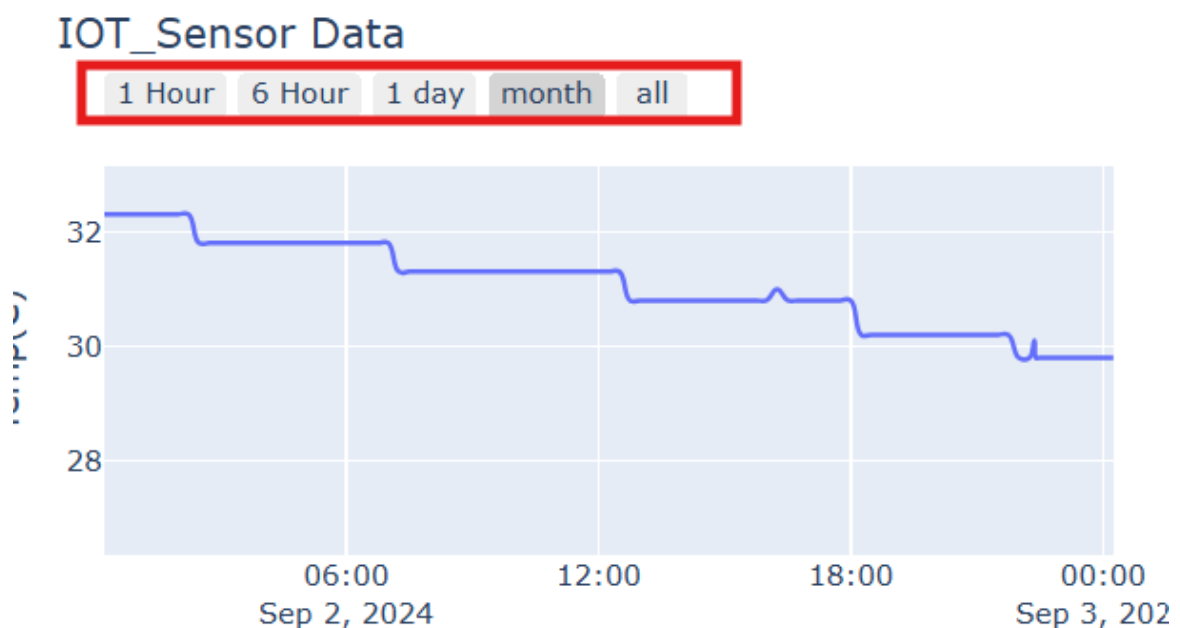
3.2.5 Dashboard Filters

Dashboard filters are tools that allow users to narrow down the data displayed on a dashboard to specific subsets or categories.

Common types of dashboard filters include:

1. Date filters: Select specific date ranges or periods.

2. Category filters: Choose from predefined categories, such as regions, products, or departments.
3. Numeric filters: Set thresholds or ranges for numeric values, like sales or temperatures.
4. Text filters: Search for specific words or phrases within text data.
5. Dropdown filters: Select from a list of predefined options.
6. Slider filters: Adjust a slider to select a range of values.
7. Checkbox filters: Select multiple options from a list.
8. Hierarchical filters: Drill down through nested categories.



3.3 Data Analytics

Now we have Data collected we can run Data analytics on collected data with below steps

Data Collection: Gather data from various sources such as sensors, drones, satellite imagery, and weather stations.

Data Cleaning: Ensure data quality by removing errors, handling missing values, and formatting data for analysis.

Data Integration: Combine data from different sources into a unified dataset for comprehensive analysis.

Data Visualization: Use plots, charts, and maps to visualize data, facilitating easy understanding and insight generation.

Descriptive Analytics: Analyze historical data to understand trends, patterns, and correlations.

Predictive Analytics: Employ machine learning algorithms to forecast future events, such as weather patterns, crop yields, and disease outbreaks.

Prescriptive Analytics: Provide actionable recommendations based on analysis, such as optimal irrigation schedules or fertilizer application rates.

Machine Learning: Train models on datasets to enable automated decision-making and predictive capabilities.

Real-time Analytics: Analyze data as it's generated to enable prompt decision-making and response to changing conditions.

Data-Driven Decision Making: Use insights generated from data analytics to inform farming decisions, optimizing operations and improving outcomes.

Continuous Monitoring: Regularly collect and analyze data to ensure ongoing optimization and improvement.

Predictive analytics on IoT data collected in smart farming can help farmers:

Predict crop yields: Using weather, soil, and crop health data to forecast yields.

Detect disease outbreaks: Identifying early signs of disease and predicting spread.

Forecast weather events: Predicting weather patterns to plan irrigation, harvesting, and other activities.

Optimize irrigation: Predicting water requirements based on soil moisture, weather, and crop water stress.

Predict equipment failures: Identifying potential equipment failures to schedule maintenance.

Analyze soil conditions: Predicting soil nutrient levels, pH, and moisture content.

Detect pests and weeds: Identifying early signs of pests and weeds to apply targeted control measures.

Predict livestock health: Monitoring animal health and predicting potential health issues.

Optimize fertilizer application: Predicting fertilizer requirements based on soil conditions and crop needs.

Predict energy consumption: Forecasting energy usage to optimize energy efficiency.

By applying predictive analytics to IoT data, smart farming can:

- Increase crop yields
- Reduce costs
- Improve resource allocation
- Enhance decision-making
- Promote sustainability

Machine learning algorithms commonly used in predictive analytics for smart farming include:

- Regression
- Decision Trees
- Random Forest
- Neural Networks
- Support Vector Machines

Chapter IV

5.1 Conclusion

Draw conclusions on the effectiveness of smart farming IoT systems for precision agriculture.

Ultimately, the goal of smart farming is to be able to observe and record data and automate processes to improve overall production output while minimising cost and preserving resources.

Smart farming technologies have the potential to transform the agricultural sector. Here are some of the advantages of smart farming.

5.1.1 Remove Human error

Using smart sensors reduces manual work, and thus human error. For example, a person taking a cow's temperature could take or record an inaccurate reading; whereas a temperature bolus inserted into the rumen of the cow sends accurate temperature data straight to a management dashboard.

5.1.2 Data collection and analysis

Data collected by smart agriculture sensors can be used to track the state of the business as well as security, staff performance, equipment efficiency and so much more.

5.1.3 More control over process

With more control (even remotely), farmers can reduce risks and plan better. For example, If you know exactly how large your yield is going to be, you can ensure that you find enough buyers and your product won't lie around unsold

5.1.4 Enhanced product quality and yield

Automation allows farmers to achieve better control over the production process and maintain higher standards of crop quality and higher yields.

5.1.5 Cost management

Increased control over production allows for better cost management. In addition, being alerted early to any anomalies in crop growth or livestock health allows farmers to mitigate costly risks.

5.1.6 Improved efficiency

Increase agricultural efficiency through process automation. Smart devices can help you automate operations such as irrigation, fertilising and pest control.

5.1.7 Reduced human resources

Because smart technology automates manual processes, this reduces your reliance on human resources. For example, instead of heading out into the fields to locate your cattle, a cattle collar connected to the IoT can report back on an animal's location data in real time.

5.1.8 Early disease detection and prevention

Lower mortality rates and ensure a healthier herd with technologies like a livestock bolus, which reports on an animal's internal temperature in real time.

5.1.9 Security

Smart sensors can detect unwanted activities happening on the farm, such as gates opening, assets being tampered with, livestock crossing geofenced

locations and more. IoT technologies can help protect the storage of crops, fertiliser and fuel, secure your farm perimeters and buildings and safeguard your workers.

Bibliography

<https://aws.amazon.com/blogs/compute/building-an-aws-iot-core-device-using-aws-serverless-and-an-esp32/>

<https://plotly.com/>

<https://circuitdigest.com/microcontroller-projects/interfacing-rain-sensor-with-arduino>

<https://how2electronics.com/connecting-esp32-to-amazon-aws-iot-core-using-mqtt/>

<https://www.arduino.cc/en/software>

<https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>

<https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html>

<https://www.elprocus.com/soil-moisture-sensor-working-and-applications/>