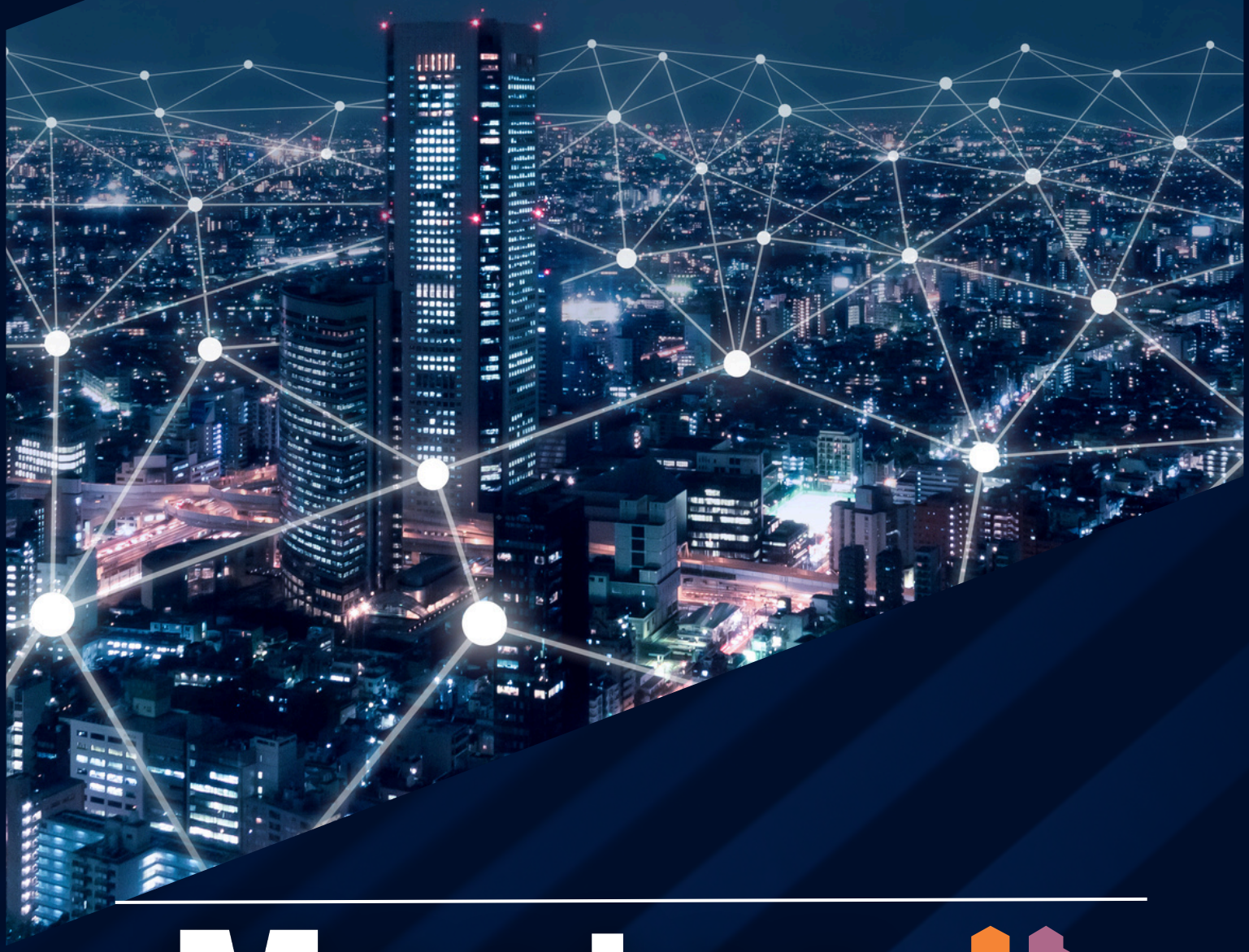
ByteWise Educators



Master Trainer

Developed by Muhammad Khair & Muhammad Afandi

MODULE 1

Smart Security System

Introduction

In an era where security and real-time monitoring are paramount, leveraging the Internet of Things (IoT) and cloud computing can offer innovative solutions. This project focuses on building a smart security system using the ESP32-CAM module and Amazon Web Services (AWS). The system employs a Passive Infrared (PIR) motion sensor to detect any movement within a designated area. Upon detecting motion, the ESP32-CAM captures an image and sends it to AWS IoT Core. The image is then securely stored in AWS S3 (Simple Storage Service). Simultaneously, the system sends a notification to the user, alerting them of the detected activity. This integration provides a scalable, efficient, and cost-effective security solution suitable for homes, offices, and other properties requiring surveillance.

Components Involved

Hardware Components:

1. ESP32-CAM Module
2. PIR (Passive Infrared) Motion Sensor
3. Connecting Wires
4. Breadboard or PCB (Optional)

AWS Services:

1. AWS IoT Core
2. AWS S3 (Simple Storage Service)
3. AWS SNS (Simple Notification Service)
4. AWS Lambda

MODULE 1
Smart Security System

Steps:

To integrate your ESP32-CAM with AWS services and build the smart security system, you'll need to set up several AWS components. Below is a step-by-step guide on how to configure AWS IoT Core, S3, SNS, and Lambda functions to work together seamlessly.

1. Set Up AWS IoT Core

a. Create an AWS IoT Thing

- Navigate to AWS IoT Core Dashboard:
  - Log in to your AWS Management Console and select AWS IoT Core from the list of services.
- Create a Thing:
  - In the IoT Core dashboard, navigate to Manage > Things.
  - Click on Register a thing.
  - Choose Create a single thing.
  - Enter a name for your device (e.g., ESP32-CAM-Device).
  - Click Next.

b. Generate Security Certificates

- Create Certificates:
  - Choose Create certificate.
  - Download the following:
    - Device Certificate (.cert.pem).
    - Private Key (.private.key).
    - Public Key (.public.key).
  - Download the Root CA Certificate
- Activate the Certificate:
  - Click on Activate to activate the certificate.

c. Attach an IoT Policy

- Create a Policy:
  - Navigate to Secure > Policies.
  - Click on Create a policy.
  - Name the policy (e.g., ESP32Policy).

MODULE 1
Smart Security System

- Set the following policy document:
  - Action: iot:* or more restrictive actions like iot:Publish, iot:Subscribe, iot:Connect, iot:Receive.
  - Resource ARN: * or specify the particular resources.
  - Effect: Allow.
- Click Create.

- Attach Policy to Certificate:
  - Go back to Secure > Certificates.
  - Find your certificate and click on the three-dot menu.
  - Select Attach policy.
  - Choose the policy you just created and click Attach.

d. Attach Certificate to Thing

- Attach Certificate:
  - In the certificate details, click on Actions > Attach thing.
  - Select your thing (e.g., ESP32-CAM-Device) and click Attach.

2. Write programming using Arduino IDE

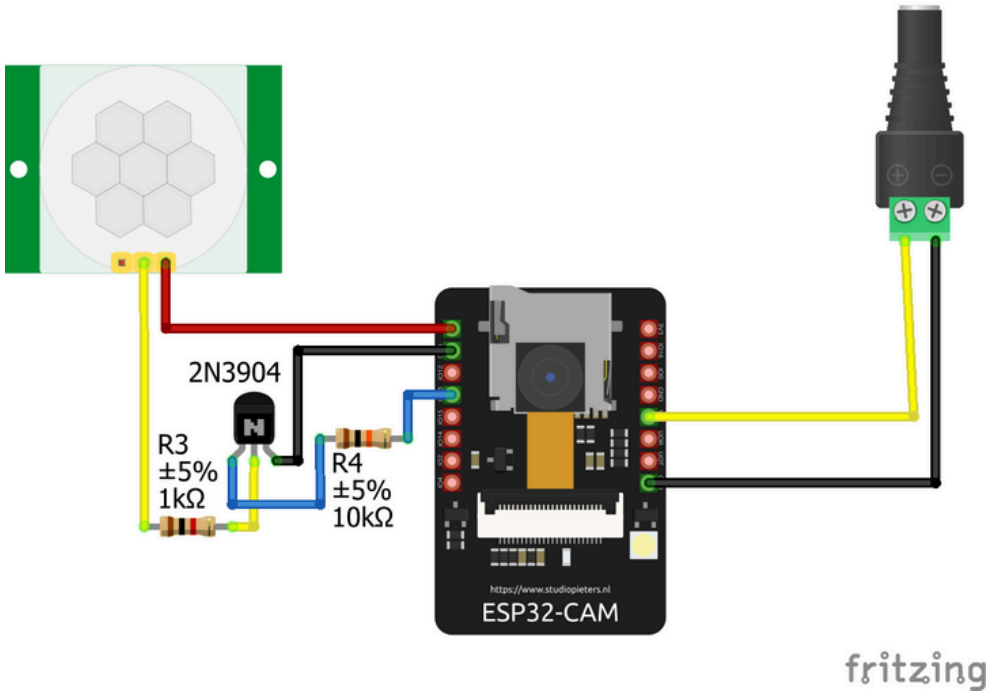
- Go to https://github.com/mdkhair/Master-Trainer/tree/main/module1
- Set the correct port in Arduino and use choose AI Thinker ESP32 cam as the board.
- Change the necessary details in the secrets.h based on your key in the certificates that you downloaded from AWS IoT Core.
- Make adjustment to PubSubClient.h

```
// MQTT_MAX_PACKET_SIZE : Maximum packet size. Override with
setBufferSize().
#ifndef MQTT_MAX_PACKET_SIZE
#define MQTT_MAX_PACKET_SIZE 1024 * 10  // Increase buffer size to 10 KB
#endif
```

MODULE 1
Smart Security System

3. Make circuit connection

- Assemble everything into one complete circuit as shown below



4. Set Up an S3 Bucket:

- Go to the AWS Management Console.
- Open the S3 service.
- Click Create bucket.
- Name the bucket (e.g., my-espcam-images).
- Choose a region, configure any settings as needed, and click Create bucket.

5. Create an AWS Lambda Function:

- Go to AWS Lambda Console:
  - In the AWS Management Console, search for Lambda and open the service.
- Create a New Lambda Function:
  - Click Create function.
  - Choose Author from scratch.
  - Give your function a name, like SaveImageToS3.

MODULE 1
Smart Security System

- Choose Python 3.x as the runtime.
- Click Create function.

6. Add S3 Write Permission to Lambda:

- In the Execution role section, click on the role name to go to the IAM role attached to this Lambda function.
- In the IAM console, click Add permissions and select Attach policies.
- Search for and select the policy AmazonS3FullAccess (or create a custom policy with permission to put objects in the specific S3 bucket).
- Attach the policy.

7. Write the Lambda Code:

- Replace the default code with the following Python code, which extracts the Base64-encoded image, decodes it, and saves it to the specified S3 bucket. Refer to my Github.

8. Create an AWS IoT Rule to Trigger Lambda:

- Go to AWS IoT Core Console:
  - In the AWS Management Console, search for IoT Core and open it.
- Create a New IoT Rule:
  - Go to Message routing > Rules in the left-hand menu.
  - Click Create.
  - Name the rule (e.g., SaveImageToS3).
  - In the Rule query statement field, write:

```
SELECT * FROM 'myespcam'
```

9. Set Up Lambda Action:

- Under Set one or more actions, choose Send a message to a Lambda function.
- Select the Lambda function you created (SaveImageToS3).
- Click Create rule.

MODULE 2

Smart Lighting Control System

Introduction

This project demonstrates how to use an ESP32 microcontroller to control a lamp using AWS IoT Core through a secure MQTT connection. The ESP32 listens to messages sent from AWS IoT, where the payload contains a relay value that controls the state of the lamp (on or off). By subscribing to an AWS IoT topic, the ESP32 can receive and process the message and act accordingly. This project highlights the integration of hardware with cloud-based services for real-time control and automation of devices.

Components Involved

Hardware Components

1. ESP32 Microcontroller
2. Relay Module
3. Lamp 240VAC (or 5VDC LED for testing)
4. Jumper Wires

2. AWS Services

1. AWS IoT Core
2. AWS IoT Thing
3. AWS IoT Policy
4. AWS Certificate Authority (CA)
5. MQTT Broker (AWS IoT)

MODULE 2

Smart Lighting Control System

Steps:

To integrate your ESP32-CAM with AWS services and build the smart security system, you'll need to set up several AWS components. Below is a step-by-step guide on how to configure AWS IoT Core, S3, SNS, and Lambda functions to work together seamlessly.

Step 1: Setting up AWS IoT

a. Create a Thing in AWS IoT Core:

1. Log in to your AWS account and navigate to the AWS IoT Core service.
2. Create a new Thing to represent your ESP32 device.
3. Download the security credentials (certificate, private key, and root CA) for your Thing. These will be used to securely connect the ESP32 to AWS IoT.

b. Create an IoT Policy

1. In the AWS IoT dashboard, create a policy that allows the ESP32 to connect, subscribe, and publish to the MQTT broker.
2. Attach the policy to your Thing.

c. Set Up MQTT Topics

- Define an MQTT topic (e.g., esp32/sub) that the ESP32 will subscribe to in order to receive the relay commands.

MODULE 2

Smart Lighting Control System

Step 2: ESP32 Configuration and Code

a. Install Arduino IDE and ESP32 Board Libraries:

1. Install the latest Arduino IDE on your computer.
2. Add the ESP32 board support by including the URL: https://dl.espressif.com/dl/package_esp32_index.json in the Arduino IDE preferences.

b. Download and Install Required Libraries:

- Install the necessary libraries like PubSubClient, ArduinoJson, and WiFiClientSecure via the Arduino Library Manager.

c. Program the ESP32:

1. Write a program (https://github.com/mdkhair/Master-Trainer/tree/main/module2) to connect the ESP32 to your Wi-Fi network and then securely connect to AWS IoT using the downloaded certificates.
2. Implement the MQTT client on the ESP32, ensuring it subscribes to the esp32/sub topic and processes messages that control the relay.

d. Upload Code to the ESP32:

- Connect the ESP32 to your computer via USB, select the appropriate board and port in the Arduino IDE, and upload the code.

Step 3: Setting up the Relay and Lamp

1. Connect the Relay to the ESP32:

- Connect the relay module to the ESP32's GPIO pin (e.g., GPIO23 for controlling the lamp).
- Connect the other side of the relay to the lamp or LED.

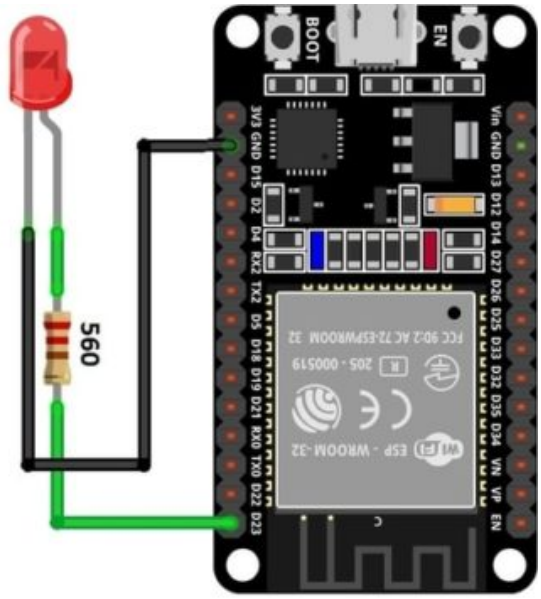
2. Power the ESP32 and Lamp:

- Ensure the ESP32 is powered via the USB or external power supply.
- Connect the lamp to an appropriate power source and ensure the relay is properly connected to handle the switching of the lamp..

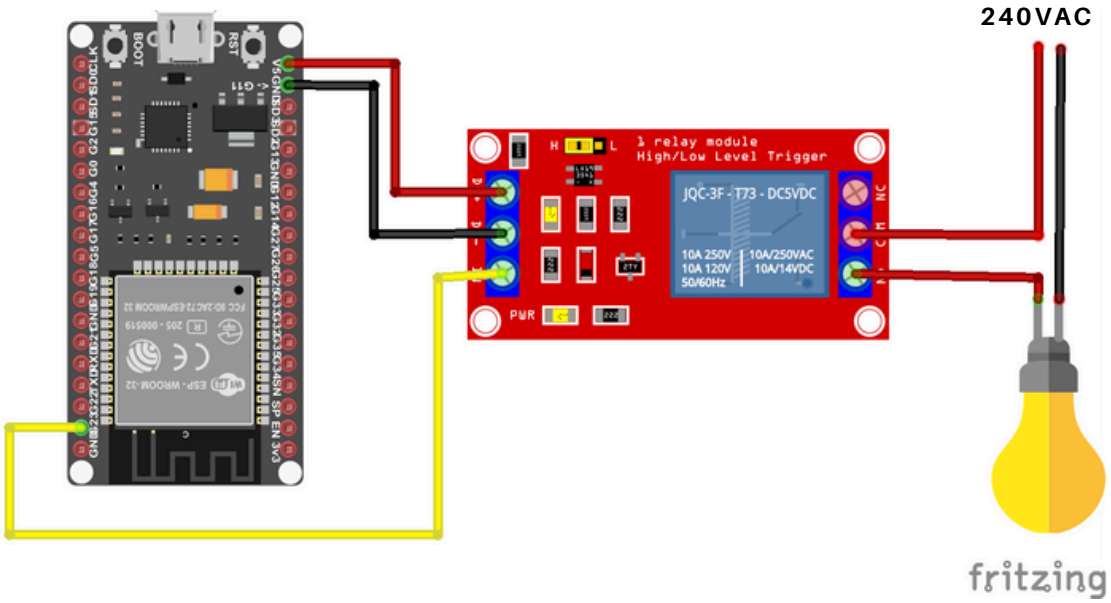
MODULE 2

Smart Lighting Control System

Circuit



For using LED for Testing



Using Relay

MODULE 2

Smart Lighting Control System

Step 4: Testing the System

a. Test MQTT Connection:

1. Monitor the serial output from the ESP32 to ensure it successfully connects to Wi-Fi and AWS IoT.

b. Send Commands from AWS IoT:

- Using AWS IoT or any MQTT testing tool (like MQTT.fx or Mosquitto), send the following JSON message to the esp32/sub topic:

```
{
 "relay": 1
}
```

- This message should turn the lamp on. Similarly, sending {"relay": 0} will turn the lamp off.

c. Monitor the ESP32:

- Check the serial output of the ESP32 to verify that it is receiving the MQTT messages and correctly changing the state of the lamp.

Conclusion

By following these steps, you can create a secure and scalable IoT application using ESP32 and AWS IoT Core to remotely control devices like lamps. This project can be extended further to integrate more sensors and actuators, enabling the development of a complete smart home or industrial automation system.

MODULE 3

Smart Energy Management

Introduction

Smart energy management systems are crucial in optimizing energy consumption and monitoring power usage in both households and industrial settings. This project aims to design a smart energy management system using the ESP32 microcontroller integrated with sensors like the DHT11 for monitoring temperature and humidity, ACS712 for measuring current, and ZMPT101B for measuring voltage. The data collected from these sensors will be sent to AWS IoT Core for real-time monitoring and analysis. AWS IoT Core provides a scalable and secure platform for managing IoT devices, and it allows users to analyze and act upon sensor data in the cloud. The goal of this project is to provide a comprehensive solution for monitoring key electrical parameters (voltage, current, and power), as well as environmental conditions (temperature and humidity), which can then be used to optimize energy consumption. The system will publish data to AWS IoT, enabling users to monitor energy usage in real-time through the cloud.

Components Involved

1. Hardware Components
   1. ESP32 Microcontroller
   2. DHT11 Sensor (Temperature and Humidity)
   3. ACS712 Current Sensor
   4. ZMPT101B Voltage Sensor
   5. Resistors, Breadboard, Jumper Wires

2. AWS Services
   1. AWS IoT Core
   2. AWS SNS (Optional)
   3. AWS Lambda (Optional)
   4. AWS CloudWatch (Optional)

MODULE 3

Smart Energy Management

Steps

Step 1: Setup Hardware Components

- ESP32 Pin Setup:
  - Connect the DHT11 sensor to pin 4.
  - Connect the ACS712 current sensor to pin 34.
  - Connect the ZMPT101B voltage sensor to pin 35.
- Power the ESP32: Connect it to your computer or a power supply via a micro-USB cable.
- Breadboard Connections:
  - Use jumper wires to connect the sensor pins to the ESP32's digital and analog pins according to the sensors' datasheets.

Step 2: Configure AWS IoT Core

- Create an AWS IoT Thing:
  - In the AWS IoT Core console, create a new thing (device).
  - Download the security certificates (public/private keys, root CA, and device certificate).
- Attach Policies:
  - Attach an IoT policy that grants permission for the ESP32 to publish and subscribe to MQTT topics.
- MQTT Topic Setup:
  - Define the MQTT topic that the ESP32 will publish to, such as esp32/sub.
- Create a Rule (Optional):
  - You can create an AWS IoT rule to trigger additional actions (like Lambda functions, SNS notifications, or data storage) based on specific conditions from the sensor data.

MODULE 3

Smart Energy Management

Step 3: Configure ESP32 for AWS IoT

- Install Arduino IDE and ESP32 Libraries:
  - Install the necessary libraries for ESP32, PubSubClient, WiFi, and ArduinoJson in the Arduino IDE.
- Upload the Sketch:
  - Program the ESP32 using the provided code from github (https://github.com/mdkhair/Master-Trainer/tree/main/module3) that integrates DHT11, ACS712, and ZMPT101B sensors.
  - Ensure that the MQTT broker is configured to the AWS IoT endpoint, and the security certificates are correctly integrated into the code.
- Test Communication:
  - Verify the connection between the ESP32 and AWS IoT by monitoring the serial output for successful connections and data publishing.

Step 4: Sensor Data Collection and Transmission

- Data Collection:
- The ESP32 will continuously collect temperature, humidity, current, voltage, and power values from the connected sensors.
- Publish Data:
- The data will be packaged in JSON format, including a Unix timestamp for each reading, and published to the MQTT topic esp32/sub.
- Data format example:

```
{
  "timestamp": 1696005783,
  "temperature": 25.5,
  "humidity": 65.0,
  "voltage": 220.0,
  "current": 1.5,
  "power": 330.0
}
```

- Monitoring:
  - You can monitor the data being published to AWS IoT Core in real-time via the AWS IoT console or by subscribing to the MQTT topic.

MODULE 3

Smart Energy Management

Step 5: Analyze Data in AWS

- Monitor in AWS IoT:
  - Use AWS IoT Core to view the incoming messages and check sensor data in real-time.
- Set Up Alarms (Optional):
  - You can set up CloudWatch alarms or Lambda functions that will be triggered if certain thresholds (e.g., high power consumption) are crossed.
- Data Storage (Optional):
  - Store the sensor data in AWS DynamoDB or Amazon S3 for historical analysis.

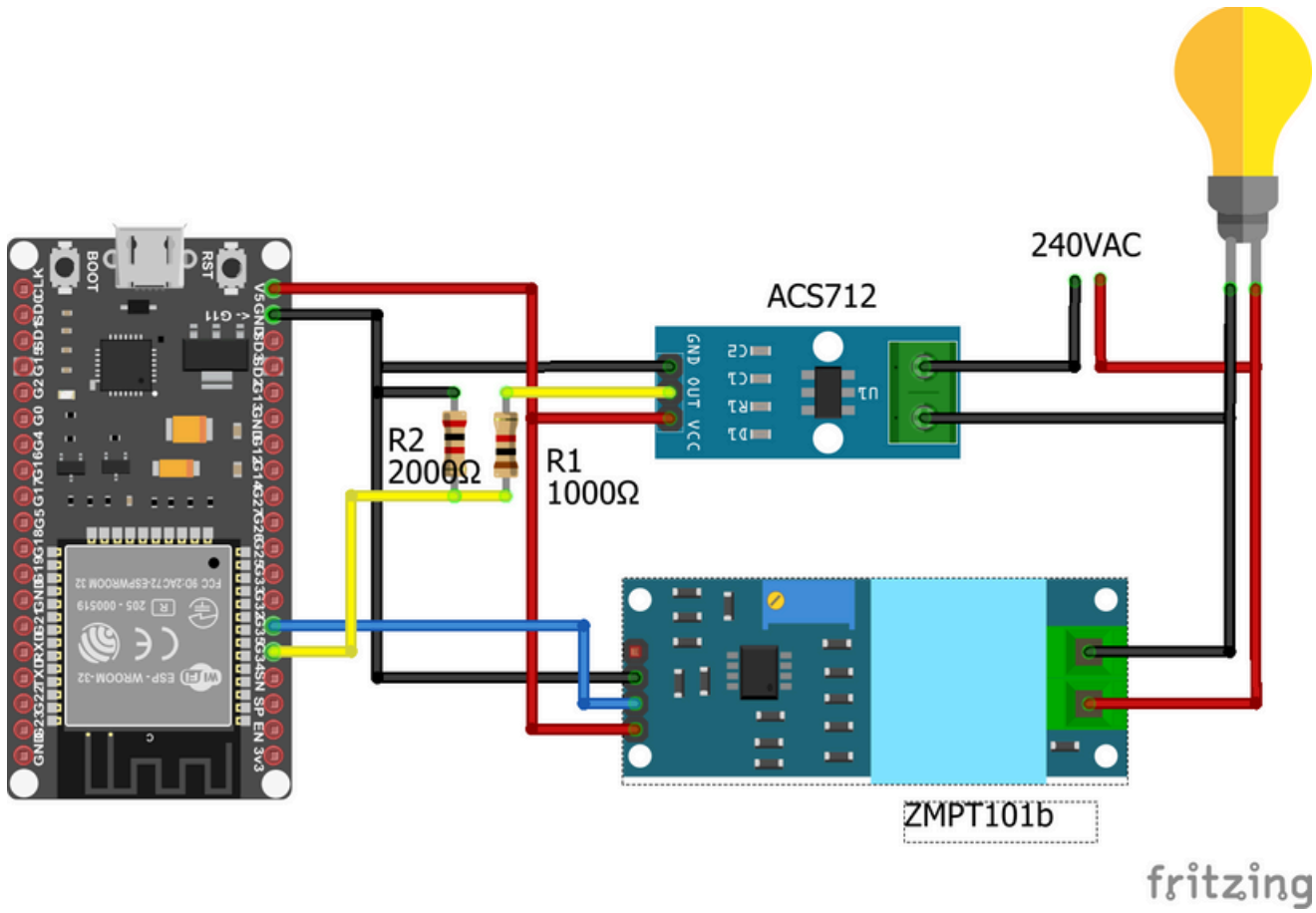
Step 6: Optional Advanced Features

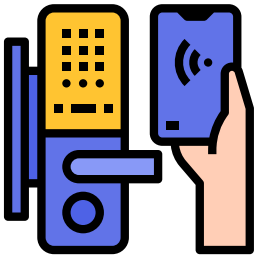
- Automation with AWS Lambda:
  - Create Lambda functions that trigger based on IoT rules, such as automatically turning off high-power devices if power consumption exceeds a certain threshold.
- Notifications with SNS:
  - Send real-time alerts via SMS or email using AWS SNS when specific energy thresholds are exceeded.
- Visualization:
  - Use AWS QuickSight or third-party tools to visualize the sensor data trends for temperature, humidity, voltage, current, and power usage.

MODULE 3

Smart Energy Management

Circuit





MODULE 4

Smart Door Lock System

Introduction

The Smart Door Lock System using ESP32, Wiegand RFID, and AWS IoT is a modern and secure access control solution designed for smart homes or any property requiring restricted entry. This system leverages RFID technology to grant or deny access based on pre-registered RFID cards. Each card is associated with a unique owner, allowing personalized access logs to be stored in the cloud. When a card is scanned, if it is recognized as authorized, a servo motor is triggered to unlock the door, and the owner's details are sent to the AWS IoT Core for monitoring and record-keeping. If an unauthorized card is detected, the system logs the event and sends a notification about the access attempt, ensuring high security and real-time alerts.

This project demonstrates how IoT (Internet of Things) technologies can be applied to real-world applications like smart door locks, enhancing both convenience and security. The integration with AWS IoT allows for seamless cloud-based storage, monitoring, and notifications, offering a scalable and flexible solution for modern access control systems.

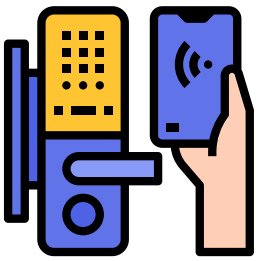
Components Involved

Hardware Components:

1. ESP32-CAM Module
2. Wiegand RFID Reader
3. RFID Cards
4. Servo motor
5. Connecting Wires
6. Breadboard or PCB (Optional)

AWS Services:

1. AWS IoT Core
2. AWS SNS (Simple Notification Service)



MODULE 4

Smart Door Lock System

Steps

1.Navigate to AWS IoT Core

- In the AWS Management Console, search for IoT Core in the search bar.
- Click on AWS IoT Core to open the service.

2.Create a New Thing

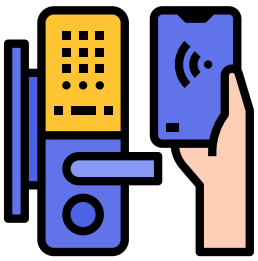
- A Thing represents your ESP32 device in AWS IoT.
  - In the AWS IoT Core dashboard, click Manage on the left panel.
  - Select Things, then click Create a Thing.
  - Choose Create a Single Thing.
  - Give your Thing a unique name (e.g., MySmartLockESP32).
  - Leave the remaining fields as default and click Next.

3. Create Certificates

- Your ESP32 will need security certificates to communicate with AWS IoT securely.
  - After naming your Thing, you will be prompted to create certificates. Click Create Certificate.
  - Download the following files:
    - Device Certificate
    - Private Key
    - Public Key
    - Amazon Root CA 1
- You can also click Attach a Policy on this page, which will lead to the next step.

4. Create and Attach a Policy

- Policies in AWS IoT define what your device is allowed to do.
  - If you didn't attach a policy in the previous step, go to the Secure section and click on Policies.
  - Click Create Policy.
  - Name your policy (e.g., ESP32Policy).
  - Under Action, enter * to allow your device to publish and subscribe:
- In the Resource ARN field, enter: *.
- For Effect, select Allow.
- Click Create to finalize the policy.



MODULE 4

Smart Door Lock System

Steps

5. Attach the Policy to the Certificate

- In the Secure section, click Certificates.
- Find the certificate you created earlier and click the ... (more options) button.
- Select Attach policy and choose the policy you created (e.g., ESP32Policy).
- Attach the policy.

6. Attach the Certificate to Your Thing

- In the Certificates section, click the ... button next to your certificate.
- Select Attach Thing and choose the Thing you created earlier (e.g., MySmartLockESP32).

7. Download Root CA and Attach to ESP32

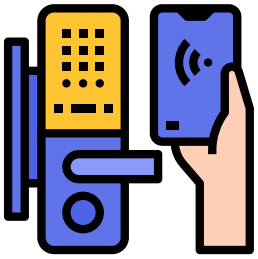
- Download the Amazon Root CA 1 certificate from the AWS IoT Device SDK page.
- Add this certificate along with your device's certificate and private key to your ESP32 code in the secrets.h file.

8. Configure MQTT Topics

- Go back to AWS IoT Core.
- In the left panel, click Test.
- Subscribe to the MQTT topic you defined in your code (e.g., myesplock).

9. Set Up AWS SNS for Notifications

- In the SNS Dashboard, on the left panel, select Topics.
- Click on the Create Topic button.
- Under Topic Name, enter a descriptive name (e.g., SmartLockNotifications).
- Under Type, choose Standard topic.
- Leave the other settings as default and click Create Topic.
- In the SNS topic page, click on Create Subscription.
- Under Protocol, choose either:
  - Email: if you want to receive notifications by email.
  - SMS: if you want to receive notifications by SMS.
- In the Endpoint field, enter your:



MODULE 4

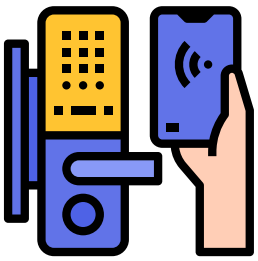
Smart Door Lock System

Steps

  - Email address: If you selected the email protocol.
  - Phone number: If you selected the SMS protocol. Ensure the phone number format is correct (e.g., +60123456789 for a Malaysian number).
- Click Create Subscription.
  - For Email: You will receive a confirmation email. Open the email and click the Confirm subscription link to complete the process.
  - For SMS: The subscription is automatically confirmed after you create it
- Go to AWS IoT Core in the AWS Management Console.
- In the left panel, select Act, then click Create a Rule.
- Rule Name: Enter a rule name (e.g., UnauthorizedCardAlertRule).
- Description: Provide a description (e.g., "This rule sends an SNS notification when an unauthorized RFID card is detected.").
- Rule Query Statement: This is where you define the condition that triggers the rule. You will query the data published by the ESP32 to AWS IoT Core. Assuming the unauthorized status is published with the key "status", the query might look like this:

```
SELECT cardID, status, CASE WHEN status = 'authorized' THEN owner ELSE 'Unauthorized Access' END AS message FROM 'myesplock'
```

- Set Up Rule Actions:
  - In the Action section, click Add Action.
  - Select Send a message as an SNS push notification.
  - Click Configure action.
- Configure SNS Action:
  - In the SNS Target dropdown, select the SNS topic you created earlier (SmartLockNotifications).
  - For Message format, choose RAW.
  - In the Message field, use the following JSON format to send the message with the card status and owner's name or "Unauthorized Access":



MODULE 4

Smart Door Lock System

Steps

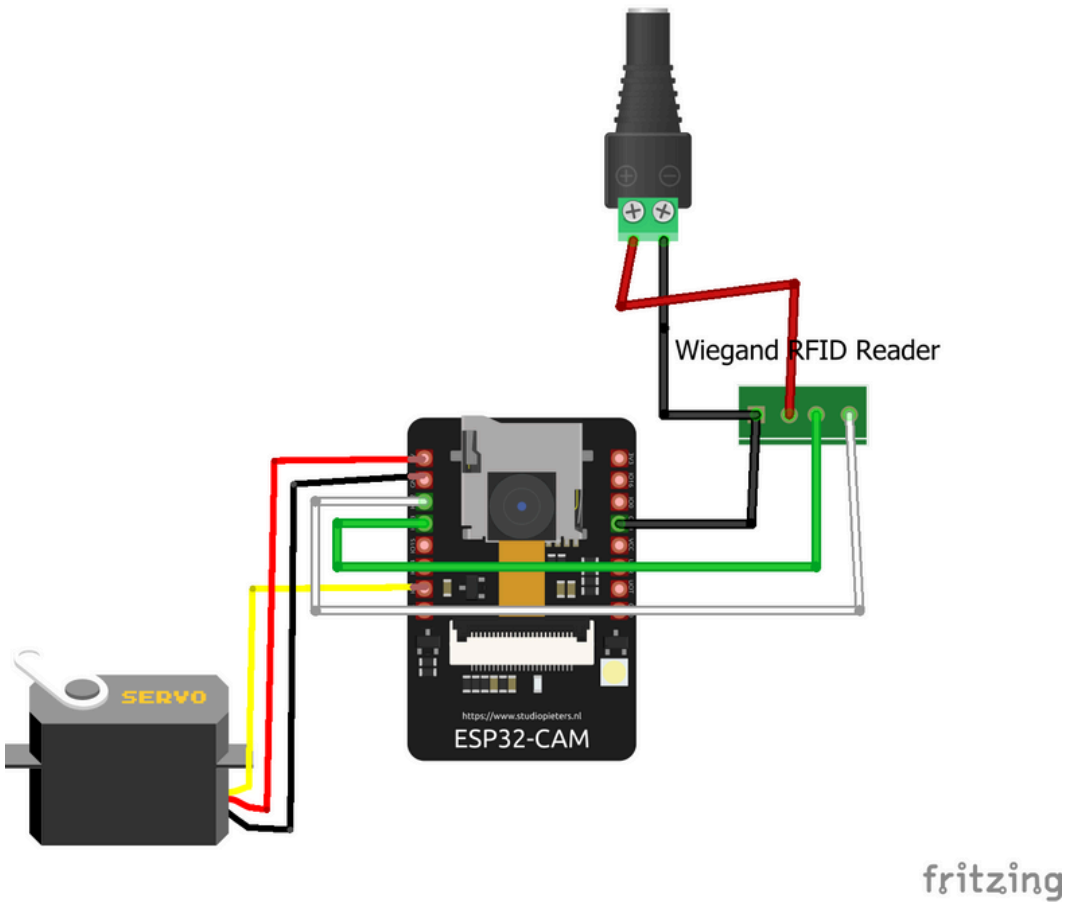
```
{
  "default": "RFID Card Detected",
  "sms": "Card Status: ${status}. Owner/Message: ${message}.",
  "email": "RFID Card Event:\n\nCard ID: ${cardID}\nStatus:
${status}\nOwner/Message: ${message}"
}
```

- Click Create Rule.

Step 12: Upload Code to ESP32

Upload the ESP32 code to your board, ensuring that your certificates and keys are correctly referenced in the secrets.h file.

Step 13: Make connection as shown below



MODULE 5

Smart Kitchen Safety System

Introduction

In recent years, the integration of smart technologies into daily household operations has gained significant traction, particularly in enhancing safety and convenience. One critical area that benefits from such innovation is the kitchen, where potential hazards such as gas leaks can pose serious risks to homeowners. To mitigate these risks, the development of a Smart Kitchen Safety System offers a proactive solution by utilizing Internet of Things (IoT) technologies to detect gas leaks and take immediate action.

This project, titled Smart Kitchen Safety System, leverages the ESP32 microcontroller and the AWS IoT Core platform to create an automated response system for detecting LPG gas leaks using an MQ6 sensor. The system is designed to monitor gas levels in real-time, triggering a series of safety measures if a leak is detected. These measures include automatically opening the kitchen window using a servo motor, turning on a 5V kitchen fan for ventilation, and sending an instant notification to the homeowner via AWS services.

Components Involved

Hardware Components:

1. ESP32 Module
2. MQ6 Gas Sensor
3. 5V fan
4. Servo motor
5. Connecting Wires
6. Breadboard or PCB (Optional)

AWS Services:

1. AWS IoT Core
2. AWS SNS (Simple Notification Service)

MODULE 5

Smart Kitchen Safety System

Step

1. Set Up an AWS Account

- Sign up for AWS if you don't have an account: Go to AWS Signup Page.
- After completing the signup, log into the AWS Management Console.

2. Create an AWS IoT Thing

- Go to AWS IoT Core: In the AWS Management Console, type "IoT Core" in the search bar and click on it.
- On the left menu, click Manage and then click Things.
- Click Create Things:
  - Select Create a single thing.
  - Give your thing a name (e.g., mykitchen_safety_system).
  - Optionally, add attributes or tags.
  - Click Next.

3.Create and Attach a Certificate

- On the next screen, click Create Certificate.
  - Download the three files: the public key, private key, and certificate.
  - Download the Root CA from this link: Amazon Root CA.
  - Save these files securely as they are required for your ESP32 to authenticate with AWS IoT Core.
- Activate the certificate: Once the certificate is created, click the Activate button.
- Click Attach a Policy to attach a permission policy.

4.Create and Attach an IoT Policy

- In the Policy section, click Create a policy.
- Define the policy as follows:
  - Policy Name: Name your policy (e.g., ESP32IoTPolicy).
  - Action: Type iot:* to allow all IoT actions, or specify actions like iot:Publish, iot:Subscribe, iot:Connect, and iot:Receive.
  - Resource ARN: Use * to allow access to all resources (or you can restrict it to specific topics if needed).
  - Effect: Allow.
- Click Create to create the policy.
- Now attach the policy to the certificate you created in Step 3 by clicking Attach.

MODULE 5

Smart Kitchen Safety System

Step

5. Configure ESP32 Sketch for AWS IoT

1. Open your Arduino IDE (or your preferred IDE) and load the ESP32 sketch.
2. In the sketch, modify the AWS IoT credentials:
    - Open the secrets.h file and enter the WiFi credentials, AWS IoT endpoint, and certificate paths.
    - Copy coding from https://github.com/mdkhair/Master-Trainer/blob/main/module5/

6. Configure AWS IoT Core for Publishing/Receiving Data

- Set up an MQTT topic:
    - In the AWS IoT Core dashboard, go to the Test section on the left sidebar.
    - Subscribe to your MQTT topic (e.g., mykitchen/gas).
- Test publishing messages from ESP32:
    - Once the ESP32 is connected to AWS IoT Core, it will publish gas levels and status (e.g., "Gas Detected" or "Gas Cleared") to the mykitchen/gas topic.
    - In the Test section, you should see messages arriving when the ESP32 sends data.

7. Create an SNS Topic

- In the AWS Management Console, search for SNS (Simple Notification Service) and select it.
- In the SNS dashboard, click Create Topic.
    - Choose Standard for the topic type.
    - Enter a Topic name (e.g., GasLeakNotification).
    - Click Create topic.

8. Subscribe to the SNS Topic

- After creating the topic, click Create subscription.
- Choose Protocol:
    - Select Email to receive notifications via email or SMS for text messages.
- Enter the Endpoint (your email or phone number).
- Click Create subscription.
    - If you selected email, check your inbox and confirm the subscription.

MODULE 5

Smart Kitchen Safety System

Step

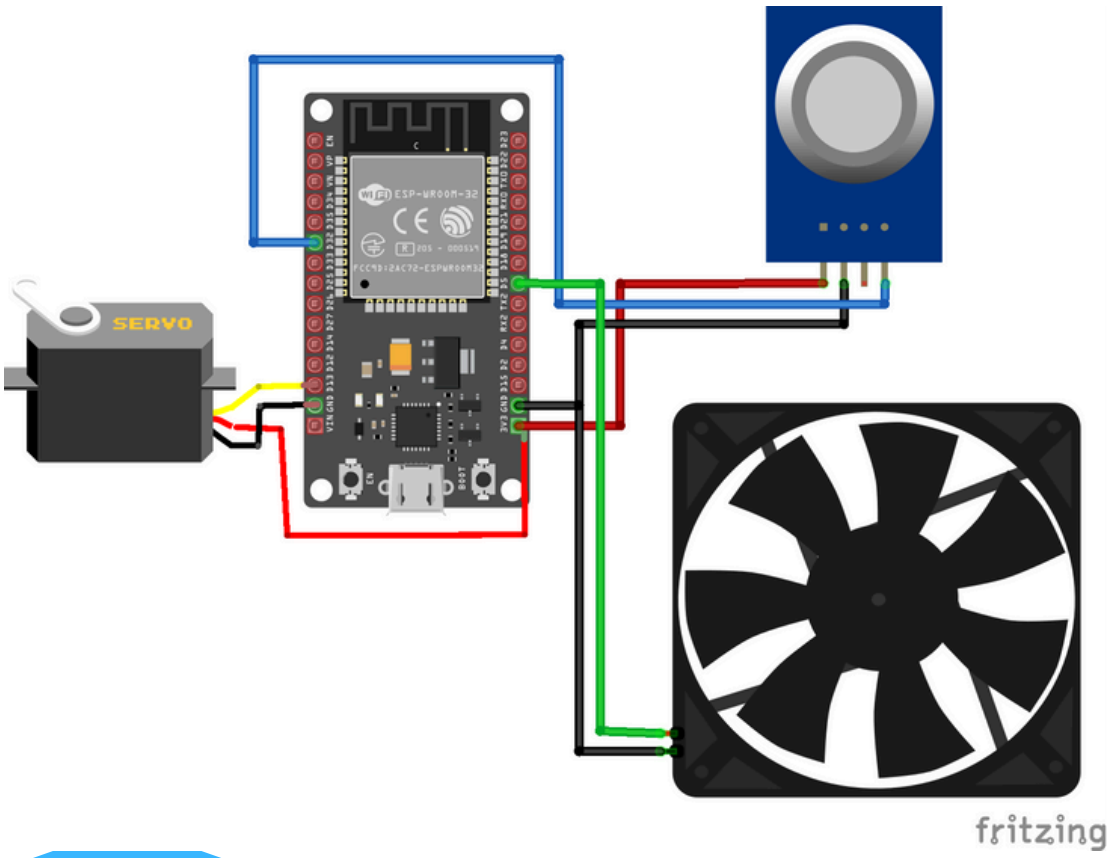
7. Set Up AWS IoT Rule for Notifications (Optional)

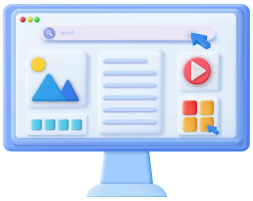
- To send a notification to your phone or email when gas is detected, you can create a rule that triggers an SNS (Simple Notification Service) message.
  - In AWS IoT Core, go to Act in the left menu and click Create a Rule.
  - Define the rule:
    - Name: Name your rule (e.g., GasDetectionRule).
    - Rule query statement: Enter an SQL-like query to filter messages from your MQTT topic

```
SELECT * FROM 'mykitchen/gas' WHERE status = "Gas Detected"
```

    - Scroll down to Set one or more actions, and click Add action.
    - Select Send a message as an SNS push notification.
    - SNS Target: Choose the SNS topic you created (GasLeakNotification).
    - Click Create to save the rule.

8. Make connection as shown below





FINAL WEB APP

Final WebApp Development with Cognito and React

Introduction

The final project combines several smart systems into a unified solution, creating a comprehensive IoT-based smart home management system. This project integrates various standalone smart systems, including Smart Security, Smart Lighting Control, Smart Energy Management, Smart Door Lock, Smart Kitchen Safety, and Smart Garden Irrigation, to enhance convenience, safety, and energy efficiency. By leveraging AWS IoT services and ESP32 microcontrollers, each subsystem communicates efficiently, allowing seamless operation and centralized monitoring through an AWS-based control panel.

ReactJS

Amazon Cognito

AWS JS SDK V2

Components Involved

a. **Hardware Components**

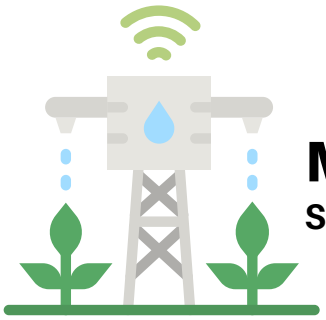
1. ESP32 Microcontroller
2. DHT11 Sensor
3. MQ6 Gas Sensor
4. PIR Motion Sensor
5. ACS712 Current Sensor
6. ZMPT101B Voltage Sensor
7. Relay Module
8. Water Pump
9. Servo Motor
10. RFID Reader (Wiegand)
11. ESP32 CAM
12. LED Lights

b. AWS Services

1. AWS IoT Core
2. AWS Lambda
3. Amazon SNS
4. Amazon S3
5. AWS Cognito
6. Amazon DynamoDB
7. Amazon CloudWatch

c. Other Components

1. Visual Studio Code
2. React
3. React Bootstrap
4. Recharts
5. React-Switch
6. AWS SDK for JavaScript
7. MQTT.js
8. SigV4Utils
9. JavaScript
10. Bootstrap CSS
11. Node.js
12. Webpack



MODULE 6

Smart Garden Irrigation System

Introduction

In recent years, the integration of smart technologies into daily household operations has gained significant traction, particularly in enhancing safety and convenience. One critical area that benefits from such innovation is the kitchen, where potential hazards such as gas leaks can pose serious risks to homeowners. To mitigate these risks, the development of a Smart Kitchen Safety System offers a proactive solution by utilizing Internet of Things (IoT) technologies to detect gas leaks and take immediate action.

This project, titled Smart Kitchen Safety System, leverages the ESP32 microcontroller and the AWS IoT Core platform to create an automated response system for detecting LPG gas leaks using an MQ6 sensor. The system is designed to monitor gas levels in real-time, triggering a series of safety measures if a leak is detected. These measures include automatically opening the kitchen window using a servo motor, turning on a 5V kitchen fan for ventilation, and sending an instant notification to the homeowner via AWS services.

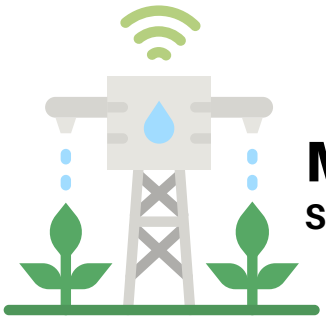
Components Involved

Hardware Components:

1. ESP32 Module
2. MQ6 Gas Sensor
3. 5V fan
4. Servo motor
5. Connecting Wires
6. Breadboard or PCB (Optional)

AWS Services:

1. AWS IoT Core
2. AWS SNS (Simple Notification Service)



MODULE 6

Smart Garden Irrigation System

Steps

1. Create an IoT Thing

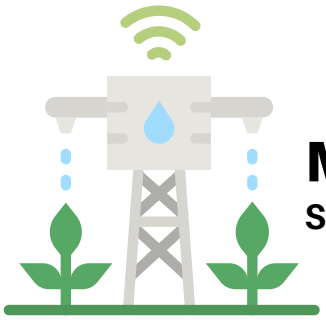
- Log in to AWS Console:
  - Navigate to the AWS Management Console.
  - Click on Services and select IoT Core under the Internet of Things category.
- Register a New Thing:
  - In the left navigation pane, click on Manage > Things.
  - Click Register a thing.
  - Choose Create a single thing.
- Configure Your Thing:
  - Name: Enter a name for your device, e.g., SmartGardenDevice.
  - Device Shadow: Enable if you plan to use shadows (optional).
  - Attributes: You can skip this section.
  - Click Next.

2.Create and Download Security Certificates

- Create Certificates:
  - On the Certificates page, select Auto-generate a new certificate (recommended).
  - Click Create certificate.
- Download Certificates and Keys:
  - Download the following files:
    - Device Certificate: xxxxxxxxxx-certificate.pem.crt
    - Private Key: xxxxxxxxxx-private.pem.key
    - Public Key: xxxxxxxxxx-public.pem.key
    - Root CA Certificate: Download Amazon Root CA 1 from here.
- Activate the Certificate:
  - After downloading, click Activate.
- Attach a Policy:
  - Click Attach a policy.
  - Since we haven't created a policy yet, we'll do that next.

3. Create an IoT Policy

- Create a New Policy:
  - In the left navigation pane, click on Secure > Policies.
  - Click Create policy.



MODULE 6

Smart Garden Irrigation System

Steps

- This policy allows your device to connect and interact with topics under smartgarden/.
- Click Create.

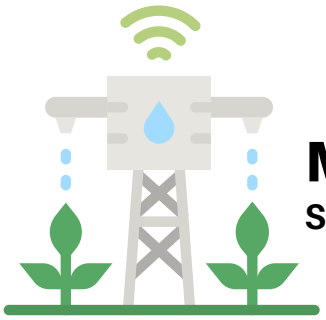
- Attach the Policy to Your Certificate:
  - Go back to Secure > Certificates.
  - Find your certificate and click the ... (more options) button.
  - Select Attach policy.
  - Choose SmartGardenPolicy and click Attach.
- Attach the Certificate to Your Thing:
  - In the same ... menu for your certificate, select Attach thing.
  - Choose SmartGardenDevice and click Attach.

4. Note Your AWS IoT Endpoint

- Find the Endpoint:
  - In the left navigation pane, click Settings.
  - Under Custom endpoint, you'll see an endpoint like xxxxxxxxxx-ats.iot.<region>.amazonaws.com.
  - Copy this endpoint; you'll need it for your ESP32 code.

5. Create an SNS Topic

- Navigate to Amazon SNS:
  - In the AWS Console, search for SNS and select Simple Notification Service.
- Create a Topic:
  - Click Topics in the left navigation pane.
  - Click Create topic.
  - Topic Type: Choose Standard.
  - Name: SmartGardenNotifications.
  - Click Create topic.



MODULE 6

Smart Garden Irrigation System

Steps

6. Subscribe to the SNS Topic

- Create a Subscription:
  - On the topic's page, click Create subscription.
  - Protocol: Choose Email or SMS.
  - Endpoint: Enter your email address or phone number.
  - Click Create subscription.
- Confirm Subscription:
  - For email, check your inbox for a confirmation email and click the link.
  - For SMS, confirmation may not be required.

7. Navigate to IoT Rules

- In AWS IoT Core:
  - Click on Act in the left navigation pane.
  - Select Rules.

8. Create a New Rule

- Click on Create Rule:
  - Click Create.
- Configure Rule Properties:
  - Name: SoilMoistureAlertRule.
  - Description: (Optional) e.g., "Triggers SNS notification when soil moisture is low".
  - Rule query statement:

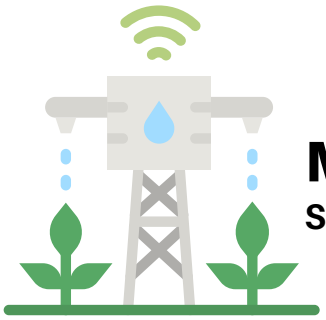
```
SELECT * FROM 'smartgarden/data' WHERE soilMoisture < 500
```

9. Set the Rule Action:

- Click Add action.
- Select Send a message as an SNS push notification.
- Click Configure action.

10. Configure SNS Action:

- SNS target ARN: Choose SmartGardenNotifications.
- Message format: Choose RAW to send the original message.
- Role name: Click Create role to allow IoT Core to publish to SNS.
  - Role name: IoTRuleToSNSRole.



MODULE 6

Smart Garden Irrigation System

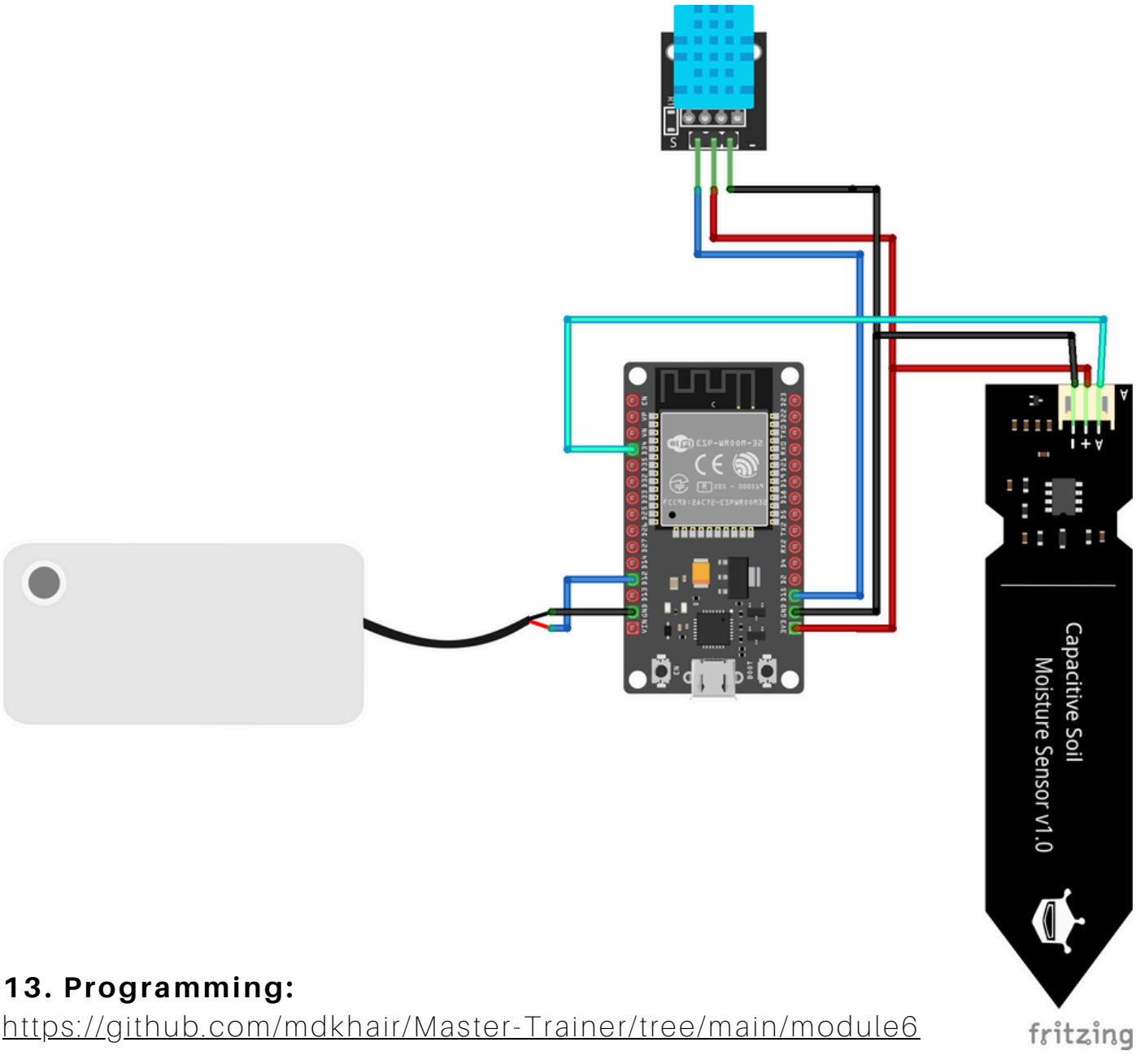
Steps

- Click Add action.

11. Create the Rule:

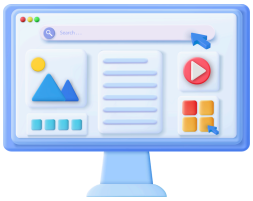
- Review and click Create rule.

12. Make connection as below



13. Programming:

https://github.com/mdkhair/Master-Trainer/tree/main/module6



FINAL WEB APP

Final WebApp Development with Cognito and React

Steps

1. Setup and Configuration:
   a. Configure AWS IoT Core for each ESP32 device to communicate securely with the cloud, using the SigV4Utils.js for signed requests.
   b. Create necessary AWS services like SNS for alerts, S3 for image storage, and DynamoDB for logging sensor data.
   c. Implement the AWS Cognito pool to manage user authentication and identity.
2. Hardware Assembly:
   a. Connect sensors (DHT11, MQ6, PIR, ACS712, ZMPT101B) and actuators (servo motors, relays) to the respective ESP32 boards.
   b. Ensure each subsystem is connected to its respective component (e.g., Smart Kitchen connects MQ6 to monitor gas levels, Smart Garden connects DHT11 for soil moisture).
3. Developing the Application:
   a. Set up a centralized control panel using React (as in App.js), where users can monitor and control all subsystems from a single interface. Display sensor data using components like charts (e.g., Recharts for visualizing DHT data).
   b. Integrate MQTT to enable real-time communication between IoT devices and the AWS IoT platform(App).
4. Subsystem Integration:
   a. Integrate each smart system into the control panel, ensuring all systems can publish data to and receive commands from AWS IoT topics. For instance, Smart Door Lock publishes RFID authentication events, and Smart Energy Management sends energy usage data to the cloud(App).
5. Testing and Optimization:
   a. Test each system independently (lighting control, security, etc.) to ensure proper communication with the AWS IoT backend.
   b. Simulate real-world scenarios (e.g., gas leakage, unauthorized door access) and monitor system responses through AWS SNS notifications and CloudWatch logs.
6. Final Deployment:
   a. Deploy the control panel on a local server or AWS EC2, allowing remote access to the home management system. Optimize sensor polling intervals and device connections to minimize latency and resource consumption.
7.