

IoT Applications (315341)

Model Question Paper — Set C · Model Answers

Q.1

a) Enabling technologies of IoT.

2 M

1. **Wireless sensor networks (WSN)** — distributed low-power sensor nodes that collect data.
2. **Cloud computing** — on-demand storage and processing of IoT data over the Internet.
3. **Big data analytics** — analysing the huge volume of IoT data to get useful information.
4. **Embedded systems** — microcontroller-based hardware/software inside every IoT device.
5. **Communication protocols** — Wi-Fi, MQTT, CoAP, LoRa, NB-IoT that connect devices.

(Any four, ½ M each)

b) UART vs I2C.

2 M

Point	UART	I2C
Type	Asynchronous (no clock line)	Synchronous (SCL clock line)
Lines	2: TX and RX	2: SDA and SCL
Devices	Point-to-point (only two devices)	Multi-device bus; 7-bit addressing, up to 112 slaves
Duplex	Full duplex	Half duplex

(Any two points, 1 M each)

c) setup() and loop().

2 M

setup() — called **once** when the sketch starts (power-on or reset). Used to initialise variables, pin modes (`pinMode`), serial communication (`Serial.begin`), Wi-Fi and libraries.

loop() — called **again and again** continuously after `setup()`. Contains the main program: reading sensors, decisions and controlling outputs.

(1 M each)

d) Use of MQ-4 sensor.

2 M

Use: MQ-4 is a semiconductor (SnO_2) gas sensor used to detect **methane (CH_4) / natural gas (CNG)** in air, about 200–10 000 ppm. It gives an analog output proportional to gas concentration and a digital alarm output.

Applications: domestic / industrial gas-leakage alarm, CNG vehicle and CNG station leak detection, coal-mine methane monitoring, IoT gas-leak alert to mobile.

(Use 1 M, two applications 1 M)

e) IR LED and photodiode in IR module.

2 M

IR LED (transmitter): continuously emits invisible infrared light ($\approx 940 \text{ nm}$) towards the front of the sensor.

Photodiode (receiver): receives the IR light reflected from an obstacle; its resistance/current changes with the received IR, and the comparator converts this into a digital output (LOW = obstacle).

(1 M each)

f) Advantages of HTTP-REST.

2 M

1. Simple and widely supported — works with every browser, server and cloud platform.
2. Stateless — server does not store client state, so it scales easily.
3. Uses standard methods (GET, POST, PUT, DELETE) and URLs; data in readable JSON/XML.
4. Firewall friendly (port 80/443) and secure with HTTPS (TLS).
5. Responses can be cached; easy to test and debug.

(Any four, ½ M each)

g) LoRa features and applications.

2 M

Features: long range (2–5 km urban, 15 km+ rural); very low power — battery life of years; low data rate 0.3–50 kbps; licence-free sub-GHz ISM band (865–867 MHz in India); chirp spread spectrum modulation.

Applications: smart metering (water/gas), smart agriculture, asset/livestock tracking, smart parking, environmental monitoring.

(Two features 1 M, two applications 1 M)

Q.2

a) Four-layer IoT architecture and difference from three-layer.

4 M

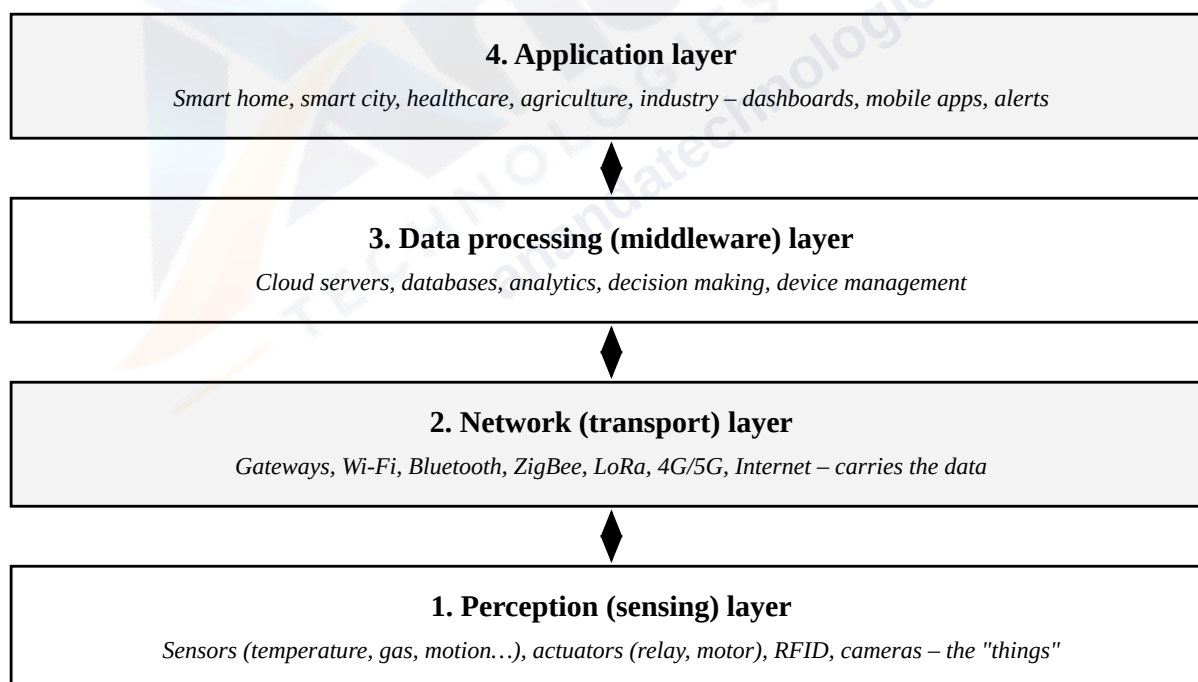


Fig.: Four-layer architecture of IoT

1. Perception layer — sensors and actuators sense and act on the physical world.

2. **Network layer** — gateways and networks (Wi-Fi, ZigBee, LoRa, 4G, Internet) carry the data.
3. **Data processing (middleware) layer** — cloud storage, analytics, decision making, device management.
4. **Application layer** — services to users: smart home, healthcare, agriculture, dashboards and apps.

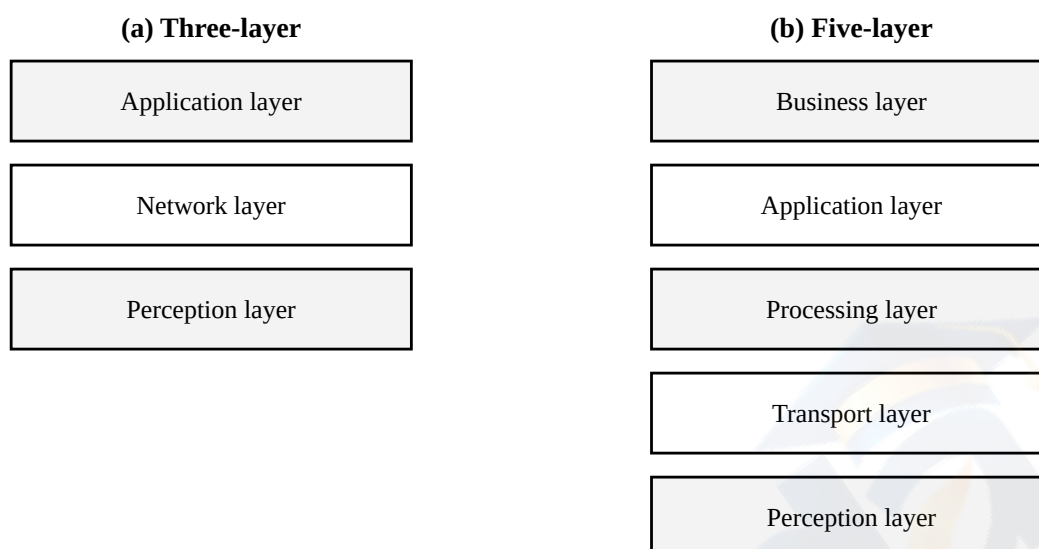


Fig.: Three-layer and five-layer IoT architectures

Difference: the three-layer model has only perception, network and application layers — data processing is hidden inside the network/application layers. The four-layer model adds a separate **processing (middleware/support) layer** for storage, analytics and security, which makes the system easier to scale and manage. (The five-layer model further splits it into transport, processing, application and **business** layers.)

(Diagram 1 M, four layers 2 M, difference 1 M)

b) Logical design of IoT.

4 M

Logical design is an abstract view of the IoT system's functions and communication, without going into hardware details.

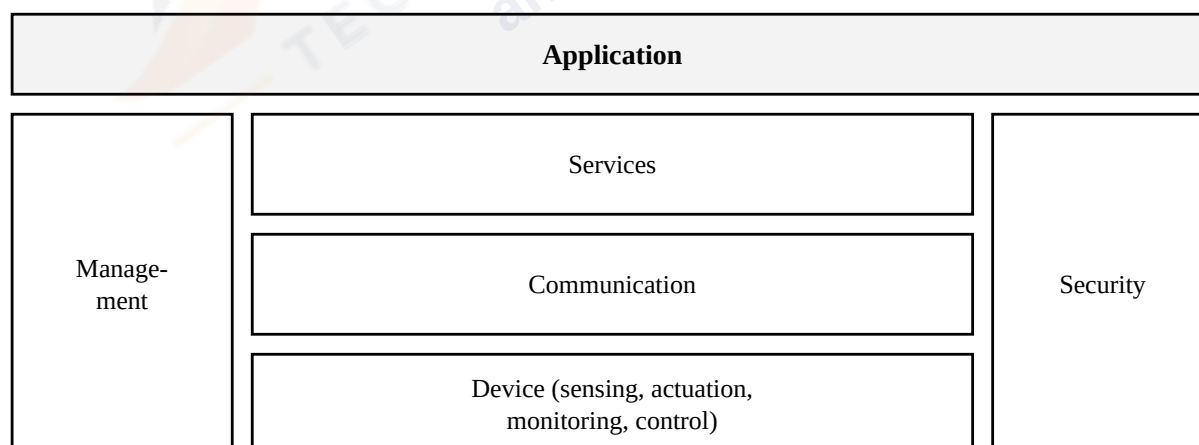


Fig.: Logical design of IoT – functional blocks

Functional blocks: **Device** (sensing, actuation, control), **Communication** (protocols), **Services** (device discovery, control, data publishing, analytics), **Management** (governance of the system), **Security** (authentication, authorisation, privacy, integrity), **Application** (user interface to monitor and control).

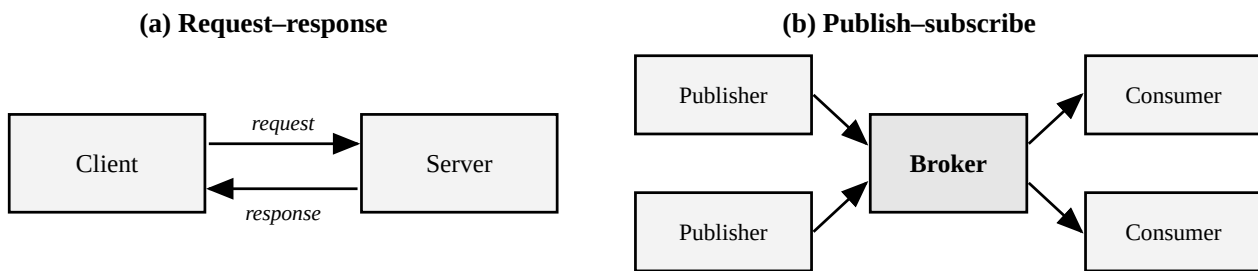


Fig.: IoT communication models – (a) request–response, (b) publish–subscribe

Communication models: **request–response** (client asks, server replies — HTTP), **publish–subscribe** (via broker — MQTT), **push–pull** (producers push data into queues, consumers pull), **exclusive pair** (full-duplex persistent connection — WebSocket). **APIs:** REST-based and WebSocket-based.

(Functional blocks 2 M, communication models 2 M)

c) NodeMCU firmware and Wi-Fi features.

4 M

Firmware is the low-level software stored in the flash memory of the ESP8266 that controls the hardware and runs the user program.

- **NodeMCU (Lua) firmware** — the original open-source firmware; programs are written in the Lua scripting language and uploaded with ESPlorer.
- **Arduino core for ESP8266** — the sketch written in Arduino IDE is compiled together with the core and libraries into a binary and flashed through the USB-UART bridge (most common in labs).
- **MicroPython** and Espressif AT-command / SDK (NONOS / RTOS) firmware can also be flashed; firmware can be updated over the air (OTA).

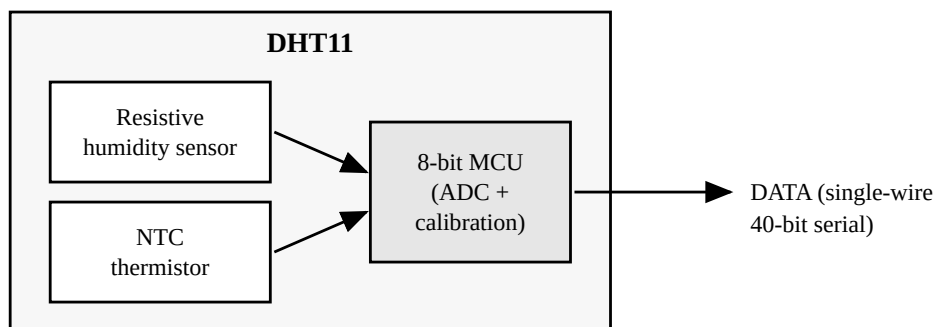
Wi-Fi features (any two):

- **Station (STA) mode** — joins an existing router, e.g. `WiFi.begin("MyHomeWiFi", "12345678")`; — then the board can send data to a cloud server.
- **Access-point (soft-AP) mode** — creates its own network, e.g. `WiFi.softAP("NodeMCU-AP")`; — phones connect directly to 192.168.4.1 (used for local configuration).
- STA+AP together; IEEE 802.11 b/g/n at 2.4 GHz; WPA/WPA2 security; built-in TCP/IP stack, so it can act as a web server or client directly.

(Firmware 2 M, two Wi-Fi features with example 2 M)

d) Working of DHT11 sensor.

4 M



Humidity: moisture-absorbing substrate between two electrodes – resistance falls as RH rises
Temperature: resistance of NTC thermistor falls as temperature rises

Fig.: Inside DHT11 – resistive humidity sensor, NTC thermistor and 8-bit MCU

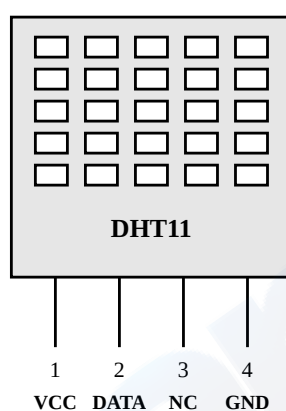


Fig.: DHT11 sensor pin diagram

- DHT11 is a low-cost digital **temperature and humidity** sensor with a calibrated digital output.
- **Humidity** is sensed by a resistive-type element: a moisture-absorbing substrate between two electrodes; as relative humidity increases the substrate absorbs water and its resistance decreases.
- **Temperature** is sensed by an **NTC thermistor** whose resistance decreases as temperature increases.
- An internal **8-bit microcontroller** measures both, applies stored calibration and sends the result on the single-wire DATA pin as a **40-bit frame** (RH integer, RH decimal, T integer, T decimal, checksum) when the host sends a start signal.
- Range 20–90 % RH (± 5 %), 0–50 °C (± 2 °C); supply 3.3–5.5 V; one reading per second.

(Diagram 1 M, working 3 M)

Q.3

a) Servo motor and PWM control.

4 M

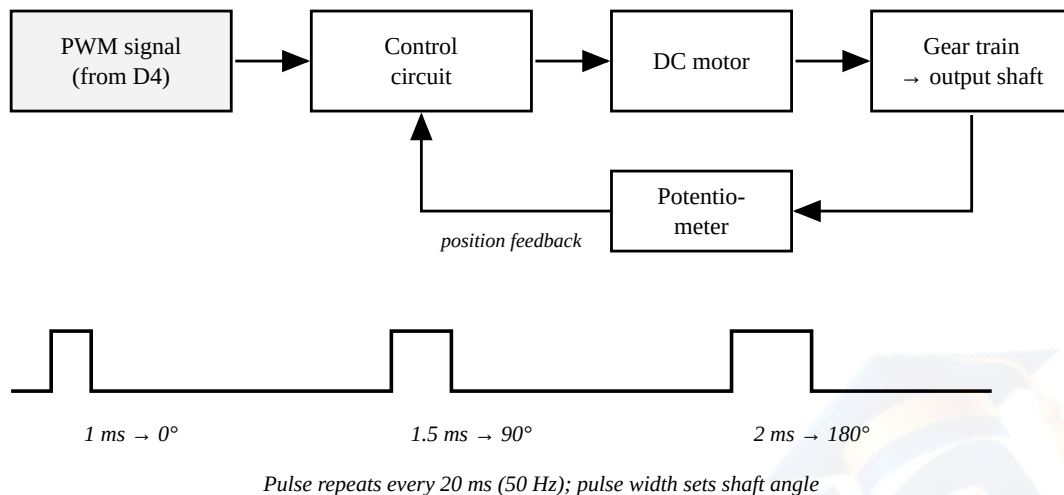


Fig.: Servo motor – internal blocks and PWM control

A **servo motor** is a rotary actuator that moves its shaft to a precise **angular position** (usually 0–180°) and holds it. It contains a DC motor, a gear train, a potentiometer (position feedback) and a control circuit — a closed-loop system.

PWM control: NodeMCU sends a pulse every 20 ms (50 Hz). The pulse width decides the angle — about 1 ms → 0°, 1.5 ms → 90°, 2 ms → 180° (exact limits differ by servo, e.g. 0.5–2.4 ms for SG90). The control circuit compares the commanded position with the potentiometer feedback and drives the motor until the error is zero.

Connections (SG90): brown = GND, red = +5 V (Vin), orange = signal → D4.

```
// Servo motor on D4 moved to 0, 90 and 180 degrees
#include <Servo.h>
Servo myServo;

void setup() {
  myServo.attach(D4, 500, 2400);    // signal pin, min/max pulse width (us)
}

void loop() {
  myServo.write(0);    delay(1000);
  myServo.write(90);   delay(1000);
  myServo.write(180);  delay(1000);
}
```

Compiled for NodeMCU (Servo library of the ESP8266 core); the shaft moves 0° → 90° → 180° every second. No Serial output.

(Servo explanation 1 M, PWM control 1 M, program 2 M)

b) LoRa wireless communication platform.

4 M

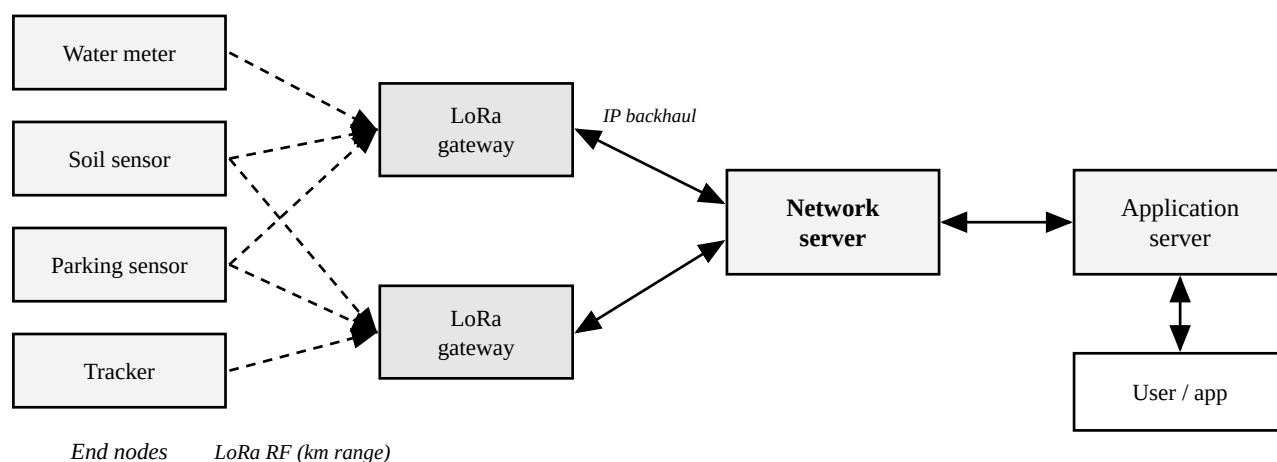


Fig.: LoRaWAN network architecture (star-of-stars)

- **LoRa** = Long Range: a physical-layer LPWAN radio technology (Semtech) based on **chirp spread spectrum (CSS)** modulation; signals can be received even below the noise level.
- **Spreading factor** SF7–SF12: higher SF → longer range but lower data rate; bandwidth 125/250/500 kHz; data rate 0.3–50 kbps.
- Works in licence-free **ISM bands** — 865–867 MHz (India), 868 MHz (Europe), 915 MHz (USA).
- **LoRaWAN** (LoRa Alliance) is the MAC/network protocol: star-of-stars — end nodes → gateways → network server → application server; AES-128 security; classes A (lowest power), B (scheduled receive), C (continuous receive).
- Range 2–5 km urban, 15 km+ rural; very low power (years on a battery); applications: smart meters, agriculture, tracking, smart city.

(Diagram 2 M, explanation 2 M)

c) MQTT topics and wildcards.

4 M

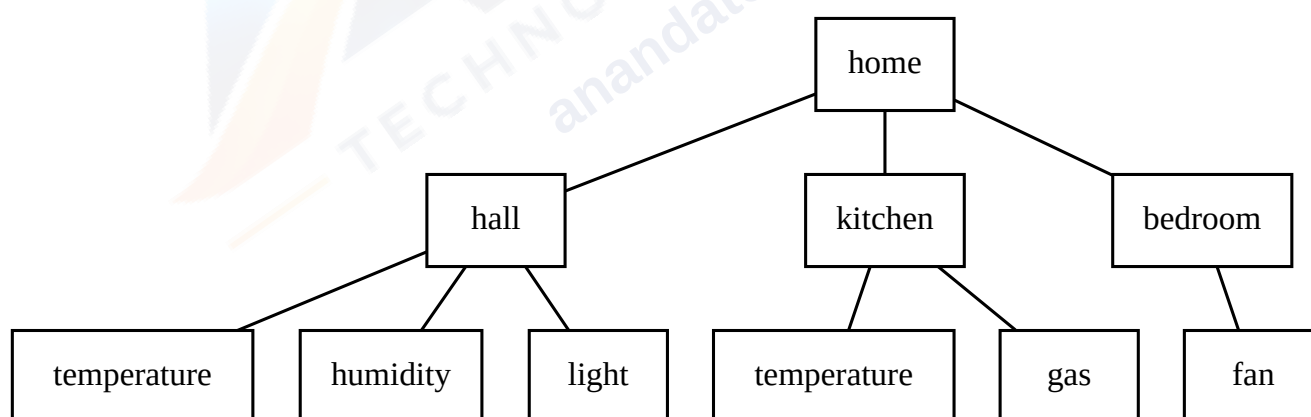


Fig.: MQTT topic hierarchy (levels separated by "/")

- A **topic** is a UTF-8 string that the broker uses to filter and route messages. It is hierarchical — **levels** are separated by a forward slash “/”, e.g. `home/kitchen/gas`.
- Topics are case-sensitive, need not be created in advance, and should not start with “/”. Topics starting with \$ (e.g. `$SYS/`) are reserved for broker statistics.

- Publishers always publish to a full topic name; subscribers may use wildcards.

Wildcard	Meaning	Subscription	Matches	Does not match
+	single level	home/+/temperature	home/hall/temperature, home/kitchen/temperature	home/hall/light
#	multi level (must be last)	home/hall/#	home/hall/temperature, home/hall/humidity, home/hall/light	home/kitchen/gas

(Topic explanation 2 M, wildcards with examples 2 M)

d) IoT based remote patient monitoring.

4 M

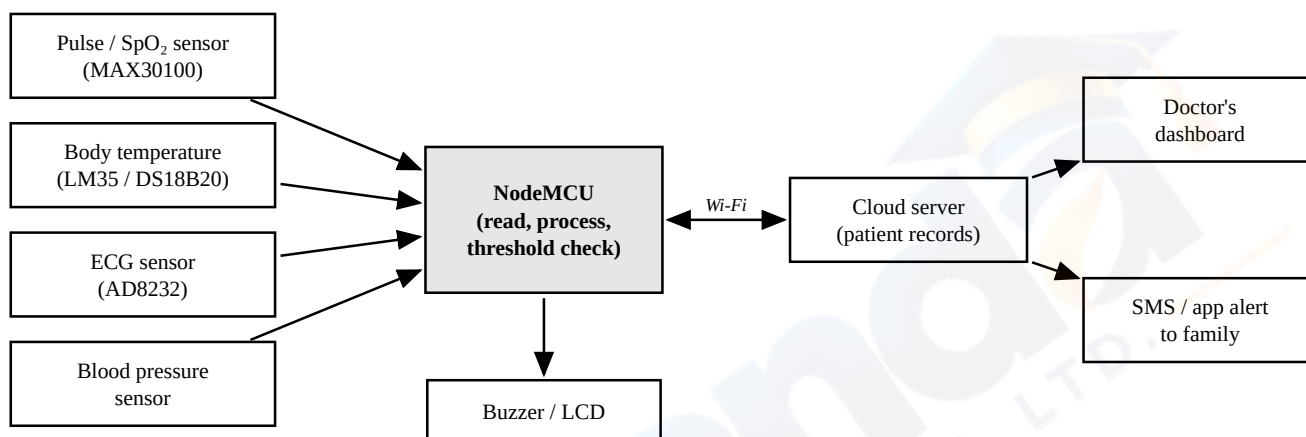


Fig.: IoT based remote patient health monitoring system

1. **Sensors** worn by the patient measure pulse rate and SpO₂ (MAX30100), body temperature (LM35/DS18B20), ECG (AD8232) and blood pressure.
2. **NodeMCU** reads the sensors, processes the values and compares them with safe limits; a local buzzer/LCD warns immediately.
3. Data is sent over **Wi-Fi** to a **cloud server** that stores the patient record and shows trends.
4. The **doctor** monitors many patients on a dashboard; abnormal values trigger **SMS / app alerts** to the doctor and family.

Benefits: continuous monitoring of elderly / home-care patients, early detection of emergencies, fewer hospital visits.

(Block diagram 2 M, explanation 2 M)

Q.4

a) PIR motion alarm program.

4 M

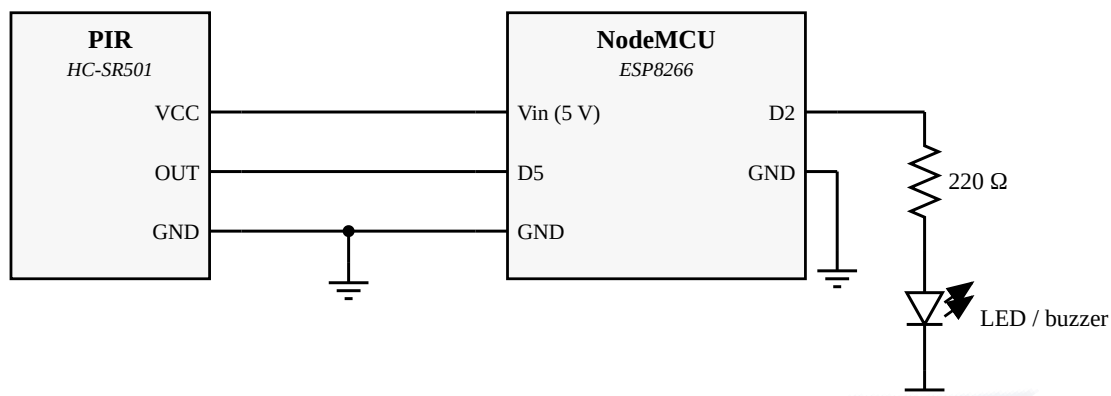


Fig.: Interfacing PIR sensor and buzzer/LED with NodeMCU

```
// PIR motion sensor on D5, buzzer/LED on D2
const int PIR = D5;
const int BUZZER = D2;
int lastState = LOW;

void setup() {
  Serial.begin(9600);
  pinMode(PIR, INPUT);
  pinMode(BUZZER, OUTPUT);
  Serial.println("PIR warming up...");
  delay(30000); // HC-SR501 needs ~30 s to settle
  Serial.println("Ready");
}

void loop() {
  int motion = digitalRead(PIR);
  digitalWrite(BUZZER, motion); // buzzer follows the PIR output
  if (motion == HIGH && lastState == LOW) {
    Serial.println("Motion detected! Buzzer ON");
  } else if (motion == LOW && lastState == HIGH) {
    Serial.println("Motion ended. Buzzer OFF");
  }
  lastState = motion;
  delay(200);
}
```

Output — Serial Monitor output (logic run on a PC with a test PIR sequence: no motion, motion, no motion, motion, no motion):

```
PIR warming up...
Ready
Motion detected! Buzzer ON
Motion ended. Buzzer OFF
Motion detected! Buzzer ON
Motion ended. Buzzer OFF
```

(Interfacing diagram 1 M, program 3 M)

b) LDR principle and voltage-divider calculation.

4 M

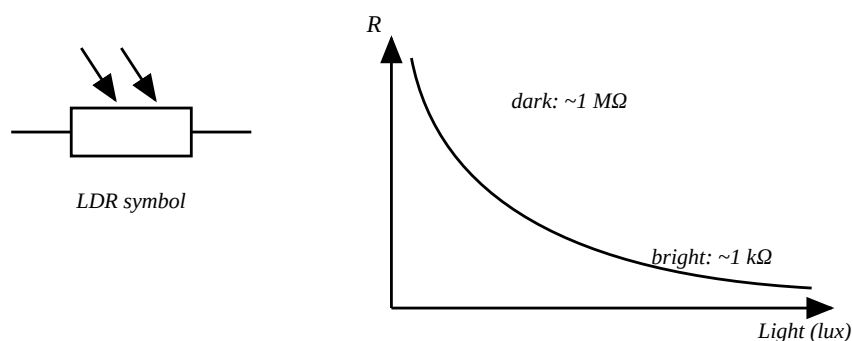


Fig.: LDR – symbol and resistance vs light intensity

Principle: an LDR (light dependent resistor / photoresistor) is made of a semiconductor such as **cadmium sulphide (CdS)**. It works on **photoconductivity**: when light falls on it, photons give energy to electrons, more charge carriers are produced and its **resistance decreases**. In darkness its resistance is very high ($\approx 1 \text{ M}\Omega$); in bright light it is low ($\approx 1 \text{ k}\Omega$ or less).

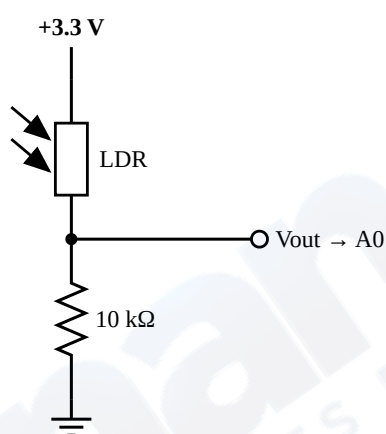


Fig.: LDR voltage divider

$$V_{\text{out}} = 3.3 \times R / (R_{\text{LDR}} + R); \text{ADC} = V_{\text{out}} / 3.3 \times 1023$$

Condition	R_{LDR}	V_{out}	ADC value
Bright light	$1 \text{ k}\Omega$	$3.3 \times 10 / (1 + 10) = 3.00 \text{ V}$	930
Darkness	$200 \text{ k}\Omega$	$3.3 \times 10 / (200 + 10) = 0.157 \text{ V}$	49

So a large ADC value means light and a small value means dark — a threshold in between (e.g. 300) is used to switch a lamp.

(Principle 2 M, calculation 2 M)

c) Smart waste management system.

4 M

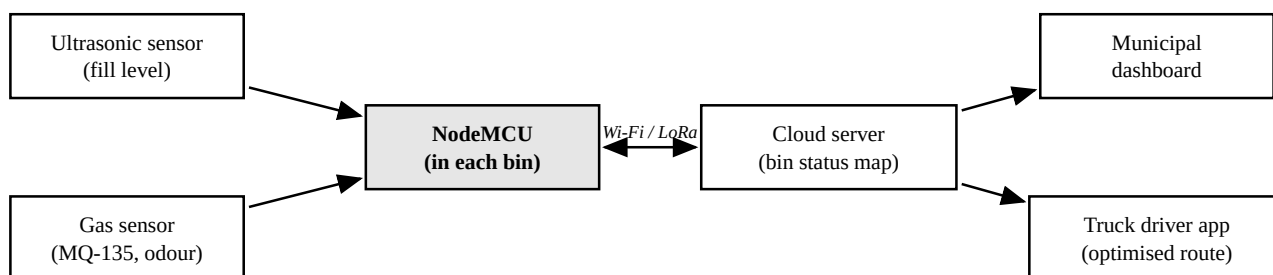


Fig.: IoT based smart waste (garbage bin) management system

1. An **ultrasonic sensor** fitted under the lid of each bin measures the distance to the garbage →
fill level (%) = $(\text{bin depth} - \text{distance}) / \text{bin depth} \times 100$.
2. A **gas sensor** (MQ-135) detects bad odour / harmful gases.
3. **NodeMCU** in each bin sends the fill level, location and gas status to the **cloud** over Wi-Fi / LoRa / GSM.
4. The municipal **dashboard** shows all bins on a map; when a bin is $\approx 80\%$ full an alert is generated and the **truck driver app** gets an optimised collection route.

Advantages: no overflowing bins, fewer truck trips (saves fuel and time), cleaner city.

(Block diagram 2 M, explanation 2 M)

d) AWS IoT and Microsoft Azure IoT.

4 M

AWS IoT (Amazon Web Services) — features:

- **AWS IoT Core:** managed message broker supporting MQTT, HTTPS and MQTT over WebSocket; scales to billions of devices.
- **Device Shadow** — stores the last reported and desired state of a device even when it is offline.
- **Rules engine** — routes data to other AWS services (S3, DynamoDB, Lambda); X.509 certificate based mutual authentication; AWS IoT Greengrass for edge computing.

Use cases: connected vehicles, industrial monitoring, smart home devices, asset tracking.

Microsoft Azure IoT — features:

- **Azure IoT Hub:** bi-directional communication between cloud and millions of devices (MQTT, AMQP, HTTPS).
- **Device twins** — JSON documents with device state and properties; device provisioning service.
- Integration with Azure Stream Analytics, Machine Learning and Power BI; **Azure IoT Edge** runs analytics on gateways; per-device authentication.

Use cases: predictive maintenance in factories, remote monitoring of equipment, smart buildings and energy management.

Note: Google Cloud IoT Core, also named in the syllabus, was retired by Google in August 2023; ThingSpeak is a free alternative used in labs.

(Each platform 2 M)

e) NodeMCU pins EN, Vin, D0, RX.

4 M

Pin	Use
i) EN	Chip enable , active HIGH. Kept HIGH (pulled up) for normal operation; pulling it LOW switches the ESP8266 off (power-down).
ii) Vin	External supply input (5 V) to the on-board 3.3 V regulator; when powered from USB it gives out the USB 5 V — used to power 5 V sensors (PIR, MQ-4, LM35).
iii) D0	GPIO16. Digital I/O; also WAKE — connected to RST to wake the board from deep sleep. No PWM, interrupt or I2C on this pin.
iv) RX	GPIO3, UART0 receive . Receives serial data from the PC (program upload, Serial Monitor) or another serial device such as GPS.

(Each pin 1 M)

Q.5

a) Greenhouse control system with flowchart.

6 M

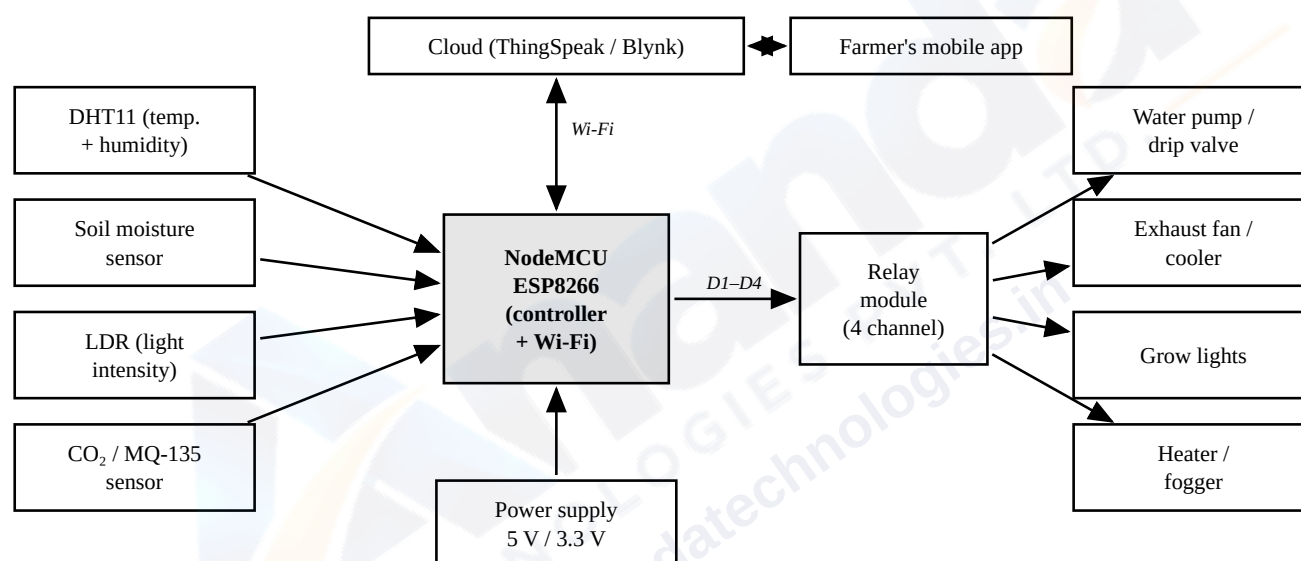


Fig.: Block diagram of IoT based greenhouse control system

Working: sensors continuously measure the greenhouse climate — DHT11 (air temperature and humidity), soil moisture sensor, LDR (light) and CO₂/gas sensor. NodeMCU compares each value with the set point and switches actuators through relays: pump/drip valve, exhaust fan/cooler, grow lights, heater/fogger. Readings are uploaded over Wi-Fi to a cloud platform (ThingSpeak/Blynk); the farmer watches graphs, gets alerts and can override actuators from the mobile app.

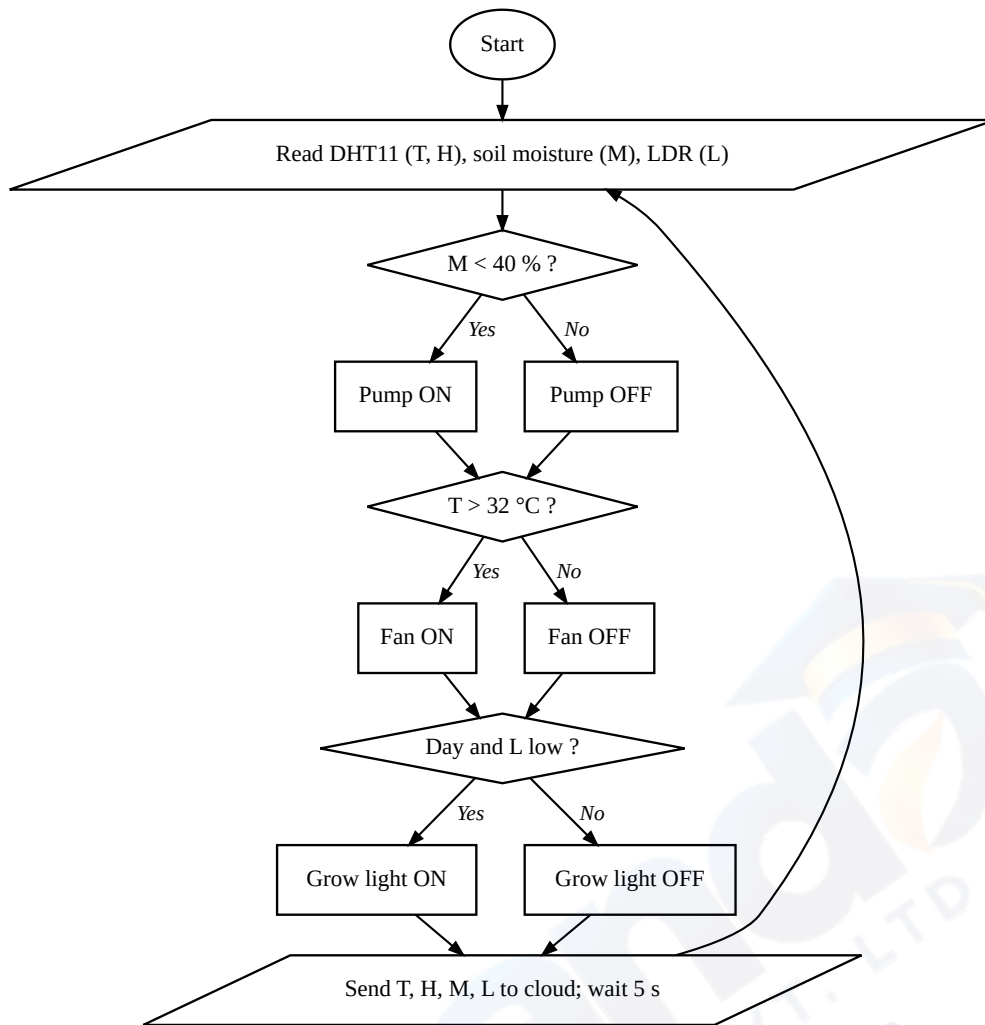


Fig.: Flowchart of greenhouse control logic (one loop)

(Block diagram 2 M, explanation 2 M, flowchart 2 M)

b) Hardware architecture of NodeMCU ESP8266.

6 M

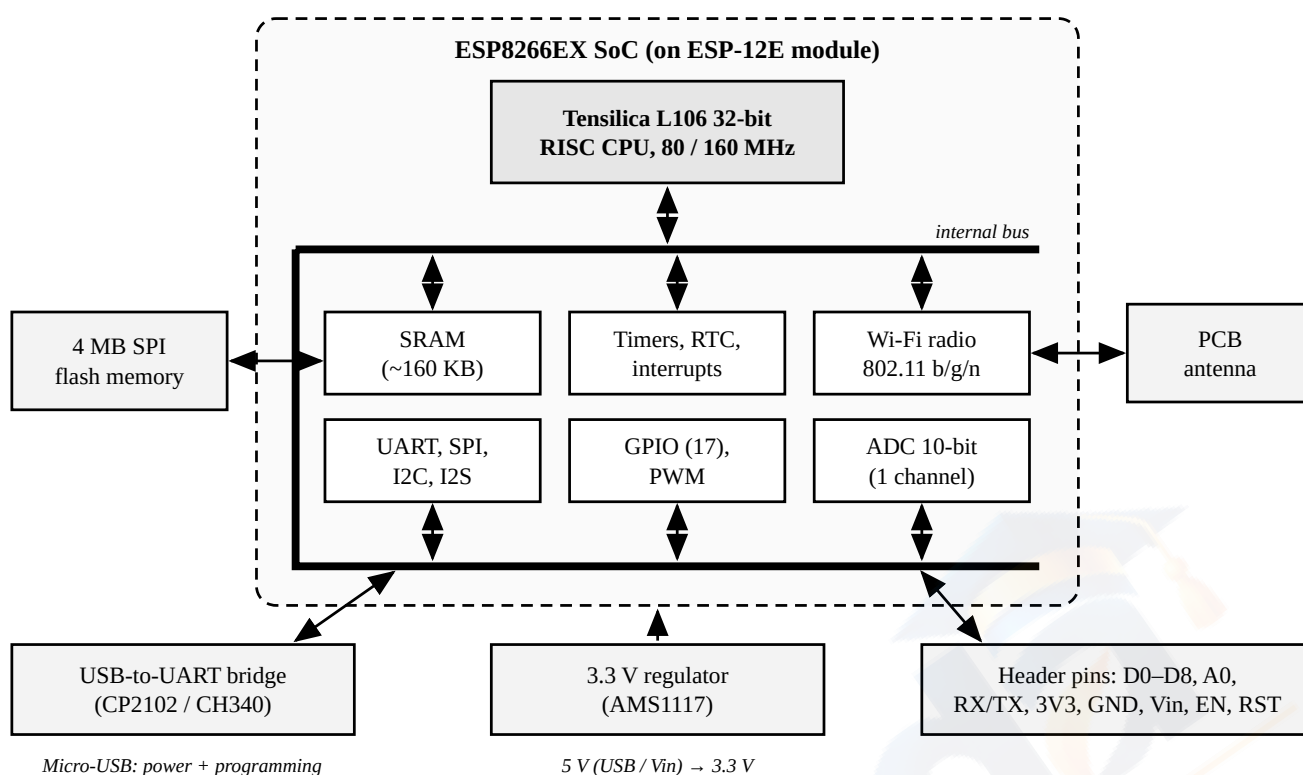


Fig.: Hardware architecture (block diagram) of NodeMCU ESP8266

1. **CPU:** Tensilica L106 32-bit RISC processor at 80 MHz (160 MHz max) — executes the program and the Wi-Fi stack.
2. **Memory:** ≈ 160 KB on-chip SRAM (instruction + data RAM); program stored in external **4 MB SPI flash** on the ESP-12E module.
3. **Wi-Fi radio:** integrated 802.11 b/g/n 2.4 GHz transceiver, power amplifier, LNA and TCP/IP stack; PCB trace antenna.
4. **Peripherals:** GPIO (17 on chip) with PWM and interrupts, 10-bit ADC (one channel, A0), UART, SPI/HSPI, I2C (software), I2S, timers and RTC.
5. **Board parts:** USB-to-UART bridge (CP2102/CH340) for programming and serial monitor, AMS1117 3.3 V regulator (input 5 V from USB/Vin), RST and FLASH push-buttons, header pins D0–D8, A0, RX/TX, 3V3, GND, Vin, EN, RST.

(Block diagram 3 M, explanation 3 M)

c) MQTT publish/subscribe program with DHT11 and relay.

6 M

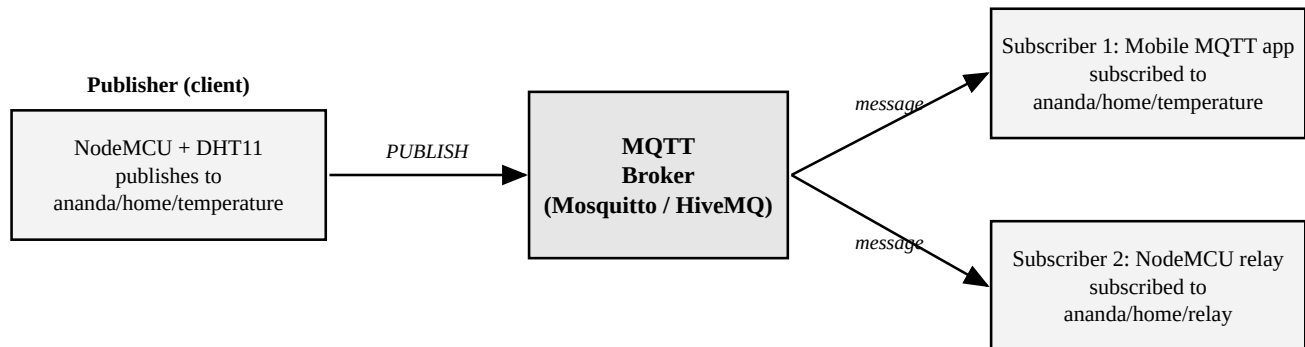


Fig.: Data flow through the MQTT broker

DHT11 DATA → D4 (10 kΩ pull-up), relay module IN → D2. Libraries: PubSubClient (MQTT) and DHT sensor library. The board publishes every 5 s and the callback() runs when a message arrives on the subscribed topic.

```

// DHT11 data to MQTT cloud broker + relay control from the cloud (NodeMCU)
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

const char* ssid = "MyHomeWiFi";
const char* password = "12345678";
const char* broker = "broker.hivemq.com"; // public test broker
const int RELAY = D2;
DHT dht(D4, DHT11);
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastSend = 0;

void callback(char* topic, byte* payload, unsigned int len) {
  String msg = "";
  for (unsigned int i = 0; i < len; i++) msg += (char)payload[i];
  Serial.print("Received ["); Serial.print(topic); Serial.print("] ");
  Serial.println(msg);
  if (msg == "ON") digitalWrite(RELAY, HIGH);
  if (msg == "OFF") digitalWrite(RELAY, LOW);
}

void reconnect() {
  while (!client.connected()) {
    if (client.connect("NodeMCU-Ananda-01")) {
      client.subscribe("ananda/home/relay"); // control topic
      Serial.println("MQTT connected");
    } else {
      delay(2000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(RELAY, OUTPUT);
  dht.begin();
}
  
```

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) delay(500);
client.setServer(broker, 1883);
client.setCallback(callback);
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();
  if (millis() - lastSend > 5000) {           // publish every 5 s
    lastSend = millis();
    float t = dht.readTemperature();
    float h = dht.readHumidity();
    if (!isnan(t) && !isnan(h)) {
      client.publish("ananda/home/temperature", String(t, 1).c_str());
      client.publish("ananda/home/humidity", String(h, 0).c_str());
      Serial.print("Published T = "); Serial.print(t); Serial.print("  H = ");
      Serial.println(h);
    }
  }
}
```

Output — Typical Serial Monitor output (depends on your network; not captured from hardware):

```
MQTT connected
Published T = 29.00  H = 61.00
Received [ananda/home/relay] ON
Published T = 29.00  H = 62.00
Received [ananda/home/relay] OFF
```

(Program 5 M (connect 1, publish 2, subscribe/callback 2), explanation 1 M)

Q.6

a) EV battery management system using IoT.

6 M

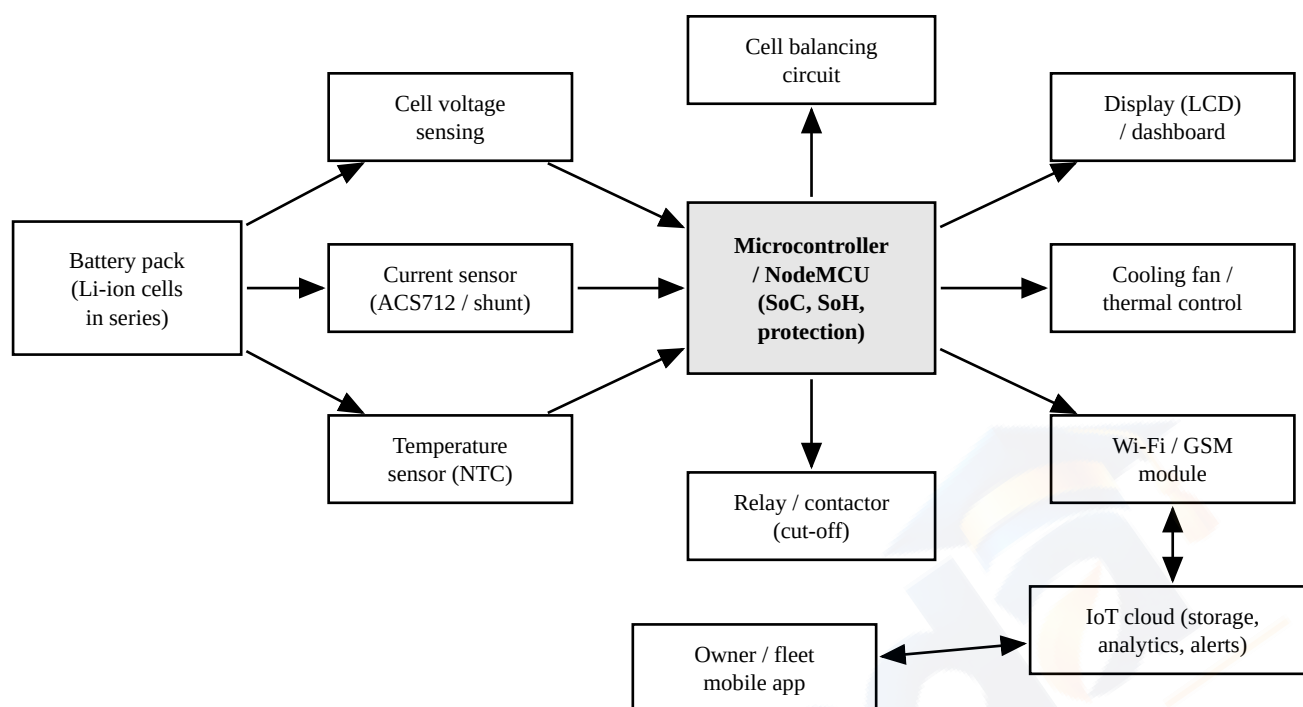


Fig.: Block diagram of EV battery management system (BMS) using IoT

Working: sensors on the battery pack measure every cell voltage, pack current and temperature. The BMS controller (microcontroller/NodeMCU) estimates the state of the battery, balances the cells and isolates the pack on a fault. The IoT module sends the data to the cloud, where the owner or fleet operator sees battery status and alerts on the mobile app, and analytics predict battery ageing.

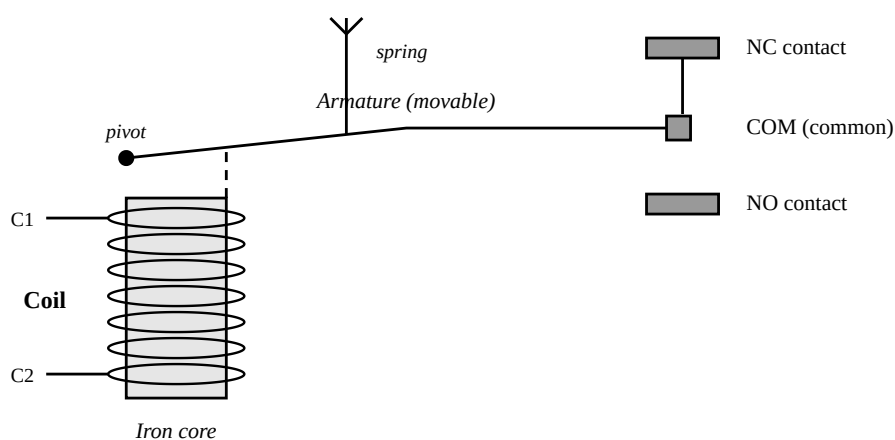
Parameters monitored	Protections provided
Cell / pack voltage	Over-voltage (over-charge) cut-off
Charge / discharge current	Under-voltage (deep discharge) cut-off
Cell temperature	Over-current and short-circuit protection
State of Charge (SoC) — remaining charge, %	Over-temperature protection — cooling fan / stop charging
State of Health (SoH) — ageing, capacity fade	Cell imbalance — cell balancing circuit

IoT features: real-time remote monitoring, range estimation, charging alerts, fault history, over-the-air updates, fleet analytics and predictive maintenance of batteries.

(Block diagram 2 M, working 1 M, parameters 1½ M, protections 1½ M)

b) Relay – construction, operation, driver circuit, applications.

6 M



Coil energised → magnetic field pulls armature → COM moves from NC to NO

Fig.: Construction of an electromechanical relay (SPDT)

Construction: coil wound on a soft-iron core (electromagnet), a movable armature on a pivot, a return spring, and contacts — COM (moving), NC (normally closed) and NO (normally open), mounted in a plastic case. Typical module: 5 V DC coil, contacts rated 10 A at 250 V AC.

Operation: coil OFF → spring holds COM on NC. Coil energised → core magnetised → armature pulled → COM switches to NO, load circuit closes. Coil OFF → spring returns armature. A low-power DC signal thus switches a high-power AC load with isolation.

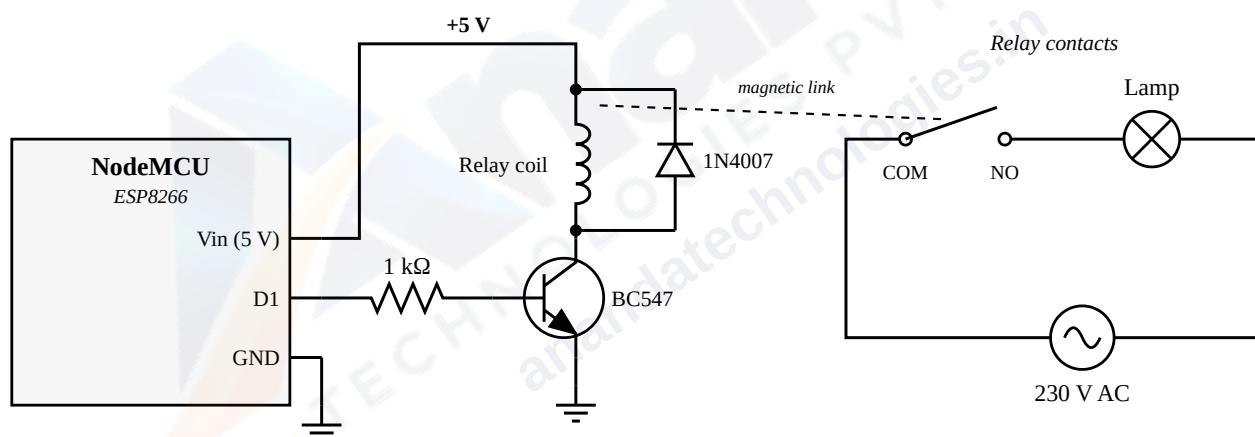


Fig.: Relay driver circuit – NodeMCU D1 switching a 230 V AC appliance

Driver: a NodeMCU pin gives only 3.3 V / ≈ 12 mA, which cannot drive the coil (≈ 70 mA at 5 V). So D1 drives an NPN transistor (BC547) through a 1 k Ω base resistor; the transistor switches the coil from 5 V. The **1N4007 flyback diode** across the coil absorbs the back-emf when the coil is switched off and protects the transistor.

Applications: home automation (lights, fans), motor and pump control, street-light switching, automatic changeover / UPS, protection relays in power systems.

(Construction 1½ M, operation 1½ M, driver circuit 2 M, applications 1 M)

c) HTTP methods and status codes.

6 M

HTTP methods (commands): GET, POST, PUT, DELETE, HEAD, PATCH, OPTIONS, CONNECT, TRACE.

Method	Explanation	REST example for an IoT device
GET	Requests (reads) a resource. Data only retrieved, no change on server; parameters in the URL.	GET /api/v1/rooms/1/temperature → 200 OK {"value": 28.6}
POST	Sends data to the server to create a new resource; data in the request body.	POST /api/v1/readings {"temp": 28.6, "hum": 60} → 201 Created
PUT	Updates / replaces the resource at the given URL with the data in the body.	PUT /api/v1/devices/fan1 {"state": "ON"} → 200 OK
DELETE	Removes the specified resource.	DELETE /api/v1/readings/1001 → 204 No Content
HEAD	Same as GET but returns only headers (no body) — used to check if a resource exists.	HEAD /api/v1/devices/fan1
PATCH	Partial update of a resource.	PATCH /api/v1/devices/fan1 {"speed": 3}
OPTIONS / TRACE / CONNECT	List supported methods / loop-back test / open a tunnel through a proxy.	—

Status codes (any two): **200 OK** — request succeeded; **201 Created** — new resource created; **400 Bad Request** — wrong request format; **401 Unauthorized** — missing/invalid API key; **404 Not Found** — resource does not exist; **500 Internal Server Error** — server failure.

(List of methods 1 M, any four explained 4 M, status codes 1 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in