IoT Applications (315341)

Model Question Paper — Set A · Model Answers

Q.1

a) IoT – definition and needs. 2 M

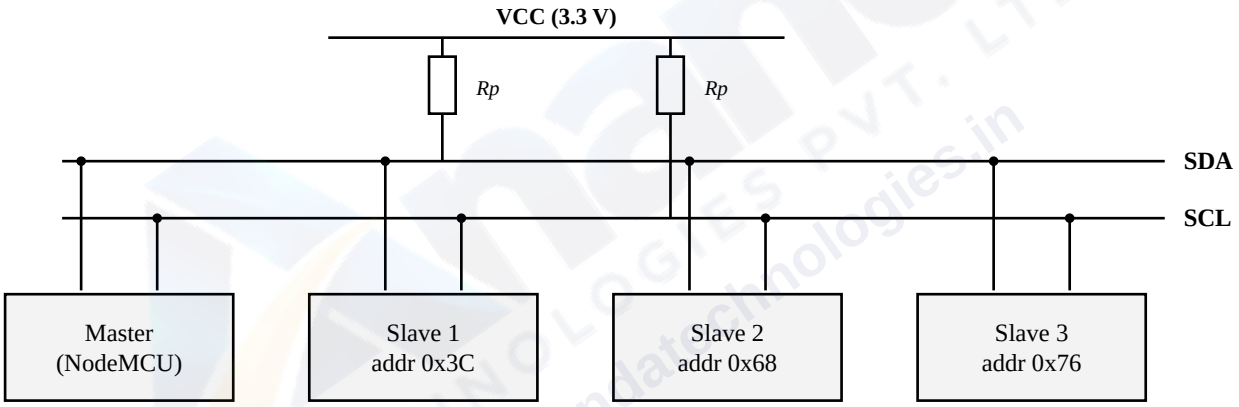
Internet of Things (IoT): IoT is a network of physical objects ("things") embedded with sensors, actuators, software and network connectivity, which collect and exchange data over the Internet and can be monitored and controlled remotely with little or no human intervention.

Needs of IoT (any two):

- Remote monitoring and control of devices (home, farm, factory) from anywhere.
- Automation — reduces human effort and errors; decisions are taken from real-time sensor data.
- Efficient use of resources such as energy, water and fuel (e.g. smart meters, smart irrigation).
- Data collection for analytics and predictive maintenance; improves safety and quality of life.

(Definition 1 M, two needs 1 M)

b) Block diagram of I2C bus. 2 M



Two open-drain lines with pull-up resistors; every slave has a unique 7-bit address

Fig.: I2C bus – one master (NodeMCU) and two or more slaves on SDA and SCL

SDA = serial data, SCL = serial clock. Both lines are open-drain and pulled up to VCC through resistors R_p. Each slave has a unique 7-bit address. On NodeMCU the default pins are SDA = D2 (GPIO4) and SCL = D1 (GPIO5).

(Diagram 2 M)

c) Features of NodeMCU ESP8266. 2 M

1. Low-cost open-source development board based on the **ESP8266** Wi-Fi SoC (ESP-12E module).
2. Built-in **Wi-Fi 802.11 b/g/n** (2.4 GHz); works as station, access point or both.
3. 32-bit Tensilica L106 CPU at 80 MHz (up to 160 MHz), 4 MB flash memory.
4. **GPIO pins** D0–D8 with PWM, I2C, SPI, UART, and one 10-bit analog input A0.

5. Programmed through micro-USB (CP2102 USB-UART) using Arduino IDE, Lua or MicroPython; 3.3 V logic.

(Any four, ½ M each)

d) Actuator. 2 M

Actuator: An actuator is a device that converts an electrical control signal (from a controller) into a physical action such as motion, switching, heat or sound. It is the output device of an IoT system — it *acts* on the environment.

Examples: relay, DC motor, servo motor, stepper motor, solenoid valve, buzzer.

(Definition 1 M, two examples 1 M)

e) Principle of IR obstacle sensor. 2 M

An IR obstacle sensor works on the principle of **reflection of infrared light**. An IR LED continuously transmits IR rays. When an object comes in front of the sensor, the rays are reflected back and fall on the IR photodiode (receiver). The photodiode resistance/current changes, and an on-board comparator (LM393) converts this into a digital output — **LOW when an obstacle is detected**, HIGH otherwise. The detection distance (about 2–30 cm) is set by a potentiometer.

(Principle 2 M)

f) HTTP methods. 2 M

Method	Use
GET	Read / retrieve a resource (e.g. sensor value)
POST	Send new data to the server (create a resource)
PUT	Update / replace an existing resource
DELETE	Delete a resource
HEAD, PATCH, OPTIONS	Headers only / partial update / list allowed methods

(Any four, ½ M each)

g) Smart city. 2 M

Smart city: A smart city is an urban area that uses IoT sensors, networks and cloud data analytics to monitor and manage city services — traffic, street lights, water, waste, energy, parking and safety — efficiently, and to improve the quality of life of citizens.

Applications (any two): smart street lighting, smart traffic control, smart parking, smart waste (bin) management, water-leak detection, air-quality monitoring, CCTV surveillance.

(Definition 1 M, two applications 1 M)

Q.2

a) Four-layer IoT architecture. 4 M

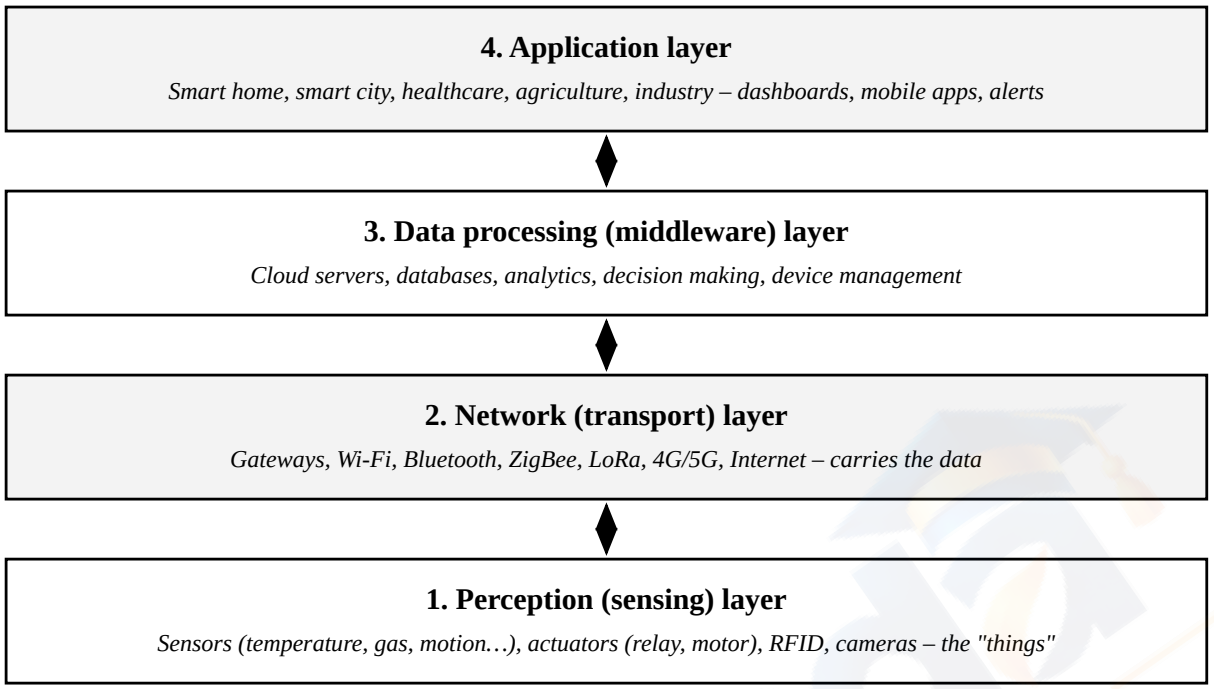


Fig.: Four-layer architecture of IoT

1. **Perception (sensing) layer:** the physical layer of “things”. Sensors sense physical quantities (temperature, gas, motion, light) and convert them to electrical data; actuators perform actions. Also RFID tags, cameras.
2. **Network (transport) layer:** transmits the sensor data from the perception layer to the processing layer, using gateways and technologies such as Wi-Fi, Bluetooth, ZigBee, LoRa, NB-IoT, 4G/5G and the Internet (IP).
3. **Data processing (middleware) layer:** stores, analyses and processes the huge amount of data received — cloud servers, databases, big-data analytics and decision making; also device management.
4. **Application layer:** delivers application-specific services to the user — dashboards, mobile apps, alerts, e.g. smart home, smart city, healthcare, smart agriculture.

(Diagram 2 M, explanation 2 M)

b) Wireless sensor network (WSN). 4 M

A **wireless sensor network** is a network of a large number of small, low-power, battery-operated **sensor nodes** spread over an area to monitor physical or environmental conditions. The nodes sense data and send it wirelessly (often hop-by-hop) to a **sink / coordinator**, which forwards it through a **gateway** to the Internet / cloud.

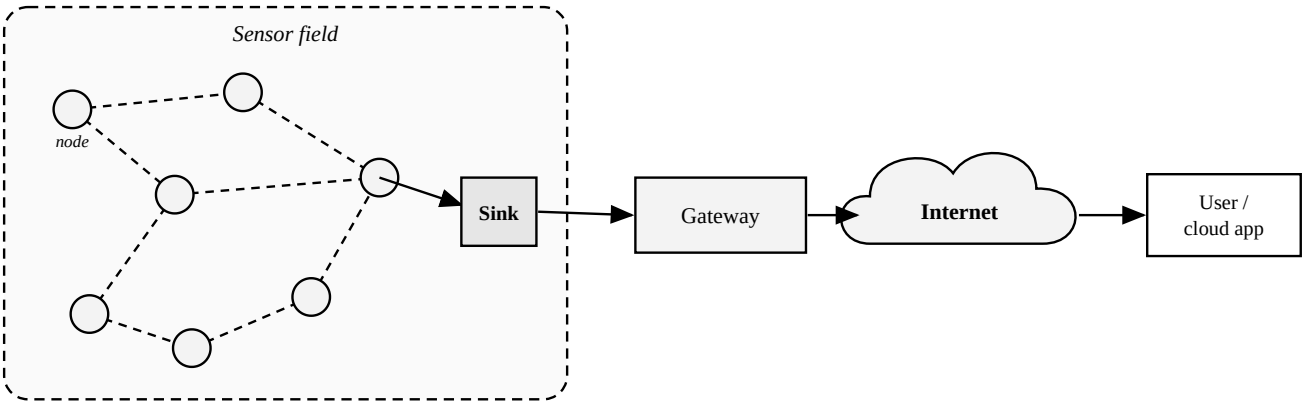


Fig.: Wireless sensor network (WSN) connected to the Internet

Components: sensor node (sensor + microcontroller + radio + battery), router nodes, coordinator/sink, gateway. Common standard: ZigBee (IEEE 802.15.4).

Why WSN enables IoT: it provides the sensing infrastructure of IoT — low-cost, low-power, self-organising data collection over a large area without wiring.

Examples: weather monitoring, smart agriculture (soil moisture), forest-fire detection, structural health monitoring, indoor air-quality monitoring.

(Explanation 2 M, diagram 1 M, examples 1 M)

c) NodeMCU pins A0, D4, 3V3 and RST.

4 M

Pin	Use
i) A0	Only **analog input** pin (ADC0). Connected to the 10-bit ADC of ESP8266 — reads 0 to 3.3 V on the board as 0 to 1023 (board has a voltage divider; the chip ADC itself is 0–1 V). Used for LDR, LM35, gas sensor analog output. `analogRead(A0)`.
ii) D4 (GPIO2)	General-purpose digital I/O pin, also UART1 TX (TXD1). The **on-board blue LED** is connected to GPIO2 and glows when the pin is LOW (active-LOW). Must be HIGH at boot. `LED_BUILTIN` = GPIO2.
iii) 3V3	**3.3 V regulated output** from the on-board AMS1117 regulator. Used to power sensors/modules (DHT11, IR module) that work on 3.3 V. ESP8266 logic level is 3.3 V.
iv) RST	**Reset** pin, active LOW. Pulling it to GND restarts the program (same as pressing the RST button). Also used with D0 (GPIO16) to wake up from deep sleep.

(Each pin 1 M)

d) Working principle of PIR sensor. 4 M

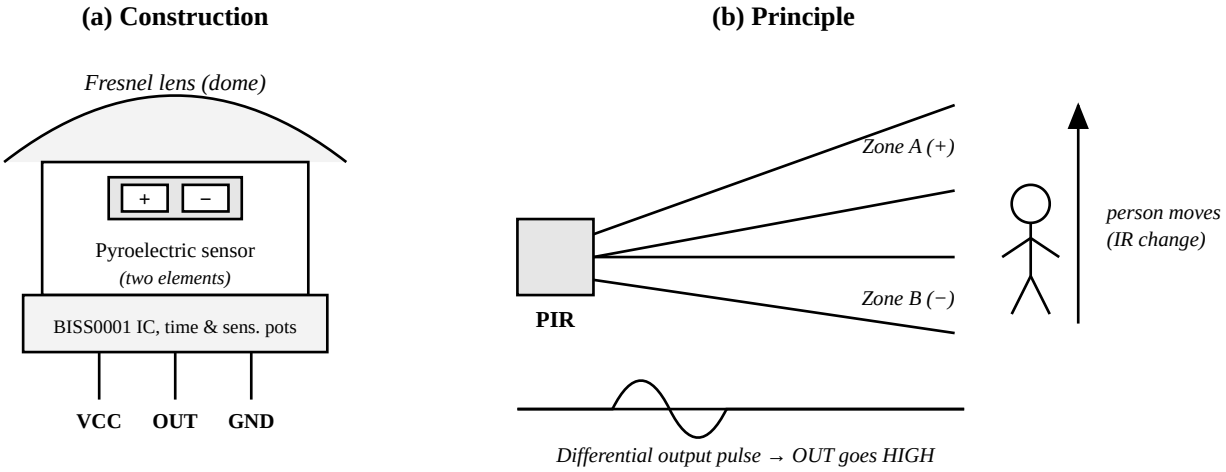


Fig.: PIR sensor – (a) construction, (b) detection principle

PIR (Passive Infra-Red) sensor detects motion of humans/animals. It is *passive* — it does not emit IR; it only senses IR (8–14 µm) radiated by warm bodies.

1. The sensing element is a **pyroelectric** crystal divided into two halves (slots) connected in opposition.
2. A **Fresnel lens** focuses IR from many zones of the room on the element and increases range (about 3–7 m, ~110° cone).
3. When nothing moves, both halves receive equal IR — outputs cancel.
4. When a person moves across, first one half and then the other receives more IR → a positive and then a negative differential pulse is produced.
5. The BISS0001 signal-processing IC amplifies this and makes the OUT pin **HIGH (3.3 V)** for a set time (time-delay pot); sensitivity pot sets range.

(Sketch 2 M, working 2 M)

Q.3

a) SPI vs I2C. 4 M

Point	SPI	I2C
Full form	Serial Peripheral Interface	Inter-Integrated Circuit
Wires	4 wires: SCLK, MOSI, MISO, SS	2 wires: SDA, SCL
Duplex	Full duplex	Half duplex
Slave selection	Separate SS (chip-select) line for each slave	By 7-bit (or 10-bit) address sent on the bus
Speed	High — several MHz (10 MHz+)	Lower — 100 kbps, 400 kbps (3.4 Mbps high-speed)
Masters	Single master	Multi-master supported
Acknowledgement	No ACK	ACK bit after every byte

Examples	SD card, TFT display, flash memory	OLED display, RTC DS1307, BMP280 sensor

(Any four points, 1 M each)

b) MQ-4 gas sensor – construction and working. 4 M

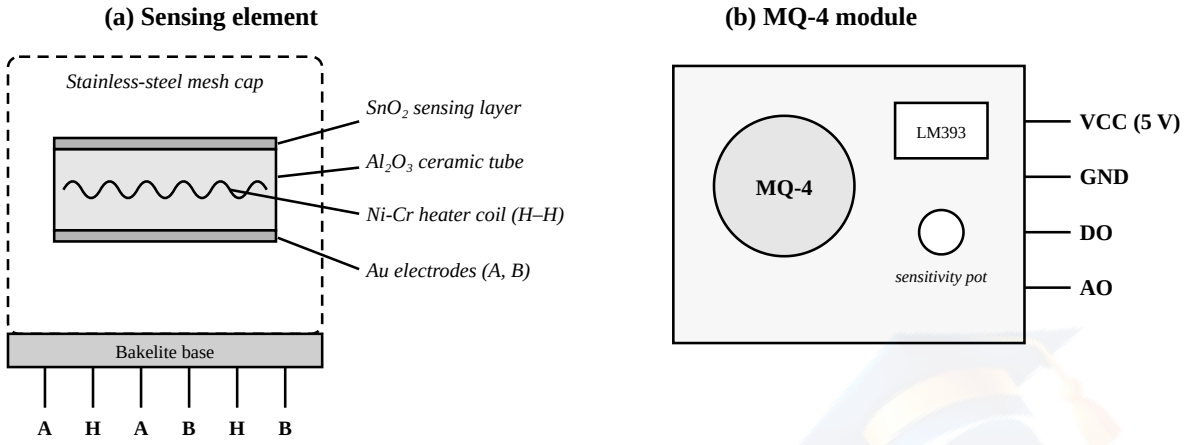


Fig.: MQ-4 gas sensor – (a) sensing element construction, (b) module pins

Construction: the sensing element is a small **Al_2O_3 ceramic tube** coated with a **tin dioxide (SnO_2)** sensitive layer. Gold measuring electrodes (A, B) touch the layer and a **Ni-Cr heater coil** (H–H) inside the tube heats it. The element is held on a Bakelite base inside a **stainless-steel mesh** (anti-explosion cap). The module adds an LM393 comparator, a sensitivity pot and pins VCC, GND, DO, AO.

Working: the heater keeps SnO_2 at a high working temperature. In clean air, oxygen is adsorbed on SnO_2 and its resistance is high. When **methane (CH_4) / natural gas** is present, the gas reacts with the adsorbed oxygen, releasing electrons, so the **sensor resistance falls** and the output voltage across the load resistor **rises** in proportion to gas concentration (about 200–10 000 ppm). AO gives this analog voltage; DO goes LOW when the level crosses the pot threshold.

(Sketch 2 M, construction 1 M, working 1 M)

c) MQTT publish–subscribe model. 4 M

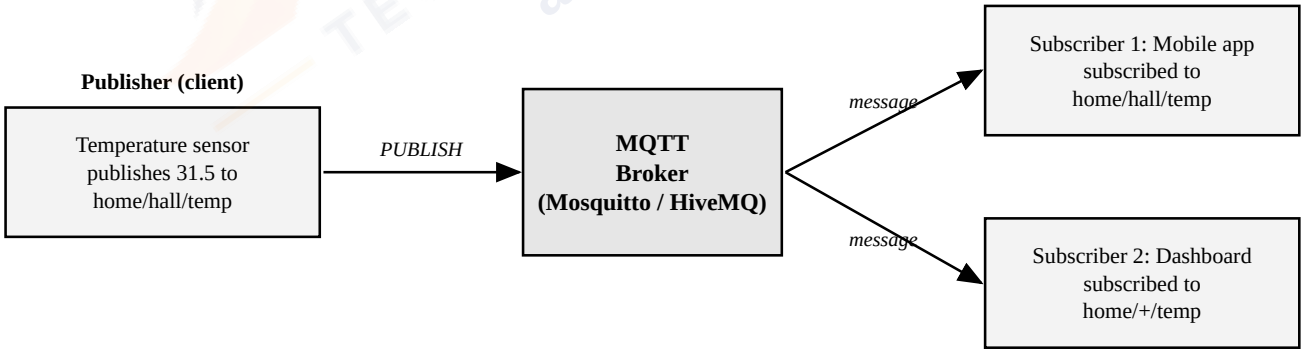


Fig.: MQTT publish–subscribe model

- **MQTT** (Message Queuing Telemetry Transport) is a lightweight publish–subscribe messaging protocol that runs over TCP/IP (port 1883, 8883 with TLS).
- **Publisher** — a client (e.g. temperature sensor node) that sends a message on a **topic**.

- **Broker** — the central server (Mosquitto, HiveMQ) that receives all messages and forwards each one to every client subscribed to that topic. Publishers and subscribers never talk directly — they are decoupled.
- **Subscriber** — a client (mobile app, dashboard, actuator node) that registers interest in a topic and receives its messages.
- Reliability is set by **QoS** levels 0 (at most once), 1 (at least once), 2 (exactly once).

Example of topic: `home/hall/temp` — the sensor publishes `31.5`; the app subscribed to `home/hall/temp` (or `home/+/temp`) receives it.

(Diagram 2 M, explanation 2 M)

d) Street light control using NodeMCU and LDR. 4 M

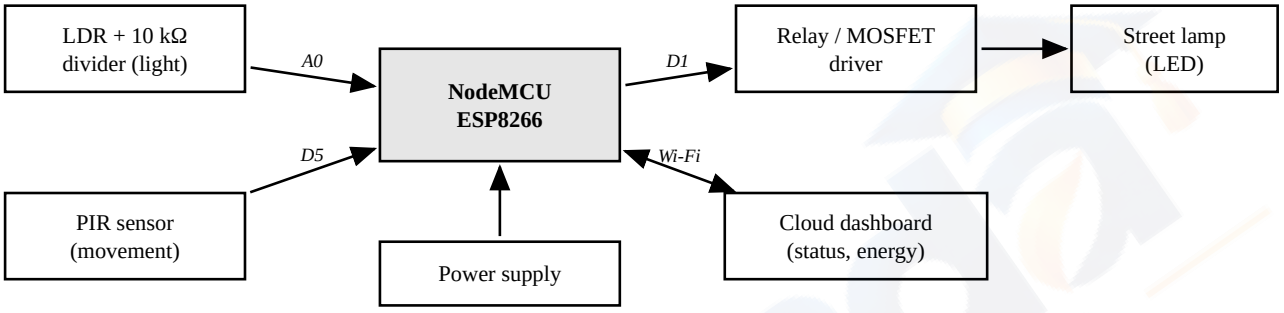
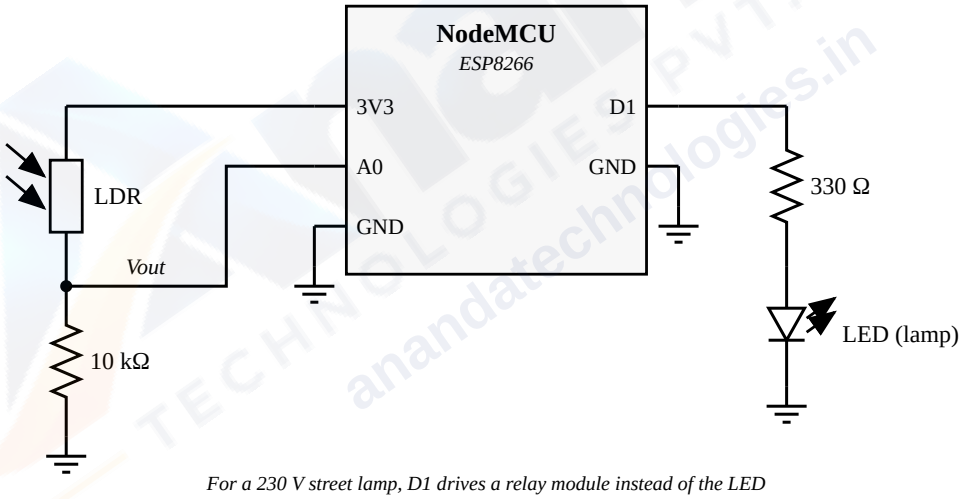


Fig.: Block diagram – street light control using LDR and NodeMCU



For a 230 V street lamp, D1 drives a relay module instead of the LED

Fig.: Circuit – LDR voltage divider on A0, lamp/LED on D1

Working: the LDR and a 10 kΩ resistor form a voltage divider connected to A0. In daylight the LDR resistance is low, so the A0 voltage (ADC value) is high; at night the LDR resistance becomes very high and the ADC value falls. NodeMCU compares the reading with a threshold: below the threshold (dark) it makes D1 HIGH, which switches ON the street lamp through a relay / MOSFET driver; in daylight D1 is LOW and the lamp is OFF. A PIR sensor can be added to dim the lamp when nobody is on the road, and the status can be sent to a cloud dashboard over Wi-Fi.

```
// Automatic street light: LDR divider on A0, lamp relay on D1
const int LAMP = D1;
const int DARK = 300;              // ADC value below which it is dark
```

```
void setup() {
  Serial.begin(9600);
  pinMode(LAMP, OUTPUT);
}

void loop() {
  int light = analogRead(A0);       // bright -> high value, dark -> low value
  Serial.print("Light level = ");
  Serial.print(light);
  if (light < DARK) {
    digitalWrite(LAMP, HIGH);
    Serial.println("  (dark)   -> Street light ON");
  } else {
    digitalWrite(LAMP, LOW);
    Serial.println("  (bright) -> Street light OFF");
  }
  delay(1000);
}
```

Output — Serial Monitor output (program logic run on a PC with test input values; on the board the values depend on the real sensor):

```
Light level = 812  (bright) -> Street light OFF
Light level = 540  (bright) -> Street light OFF
Light level = 310  (bright) -> Street light OFF
Light level = 212  (dark)   -> Street light ON
Light level = 95  (dark)   -> Street light ON
```

(Block/circuit diagram 2 M, working 2 M)

Q.4

a) LM35 temperature monitoring with fan control. 4 M

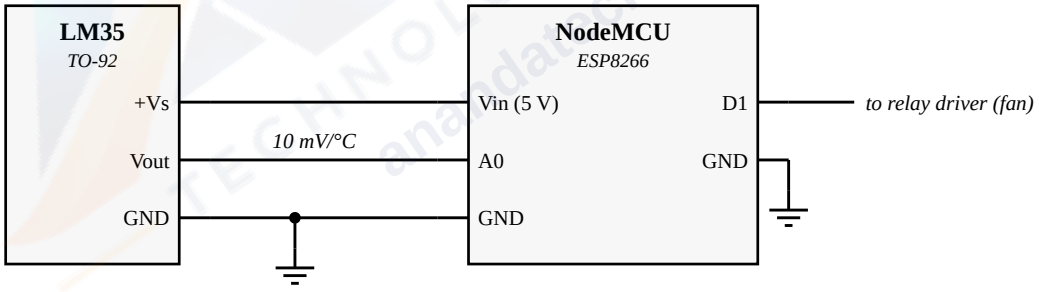


Fig.: Interfacing LM35 temperature sensor with NodeMCU

LM35 gives 10 mV/°C. On NodeMCU, A0 reads 0–3.3 V as 0–1023, so **T (°C) = ADC × (3.3 / 1023) × 100**.

```
// LM35 on A0, fan (through relay module) on D1 - NodeMCU ESP8266
const int FAN = D1;                 // relay input pin
const float LIMIT = 35.0;           // switch-on temperature in deg C

void setup() {
  Serial.begin(9600);
  pinMode(FAN, OUTPUT);
```

```
  digitalWrite(FAN, LOW);           // fan OFF at start
}

void loop() {
  int adc = analogRead(A0);                    // 0 - 1023
  float volt = adc * (3.3 / 1023.0);           // NodeMCU A0: 0 - 3.3 V
  float temp = volt * 100.0;                   // LM35: 10 mV per deg C
  Serial.print("ADC = ");  Serial.print(adc);
  Serial.print("  Temperature = ");  Serial.print(temp, 1);
  if (temp > LIMIT) {
    digitalWrite(FAN, HIGH);
    Serial.println(" C  -> Fan ON");
  } else {
    digitalWrite(FAN, LOW);
    Serial.println(" C  -> Fan OFF");
  }
  delay(2000);
}
```

Output — Serial Monitor output (program logic run on a PC with test input values; on the board the values depend on the real sensor):

```
ADC = 93  Temperature = 30.0 C  -> Fan OFF
ADC = 105  Temperature = 33.9 C  -> Fan OFF
ADC = 112  Temperature = 36.1 C  -> Fan ON
ADC = 120  Temperature = 38.7 C  -> Fan ON
ADC = 101  Temperature = 32.6 C  -> Fan OFF
```

(Circuit/logic 1 M, program 3 M)

b) LoRaWAN architecture and applications. 4 M

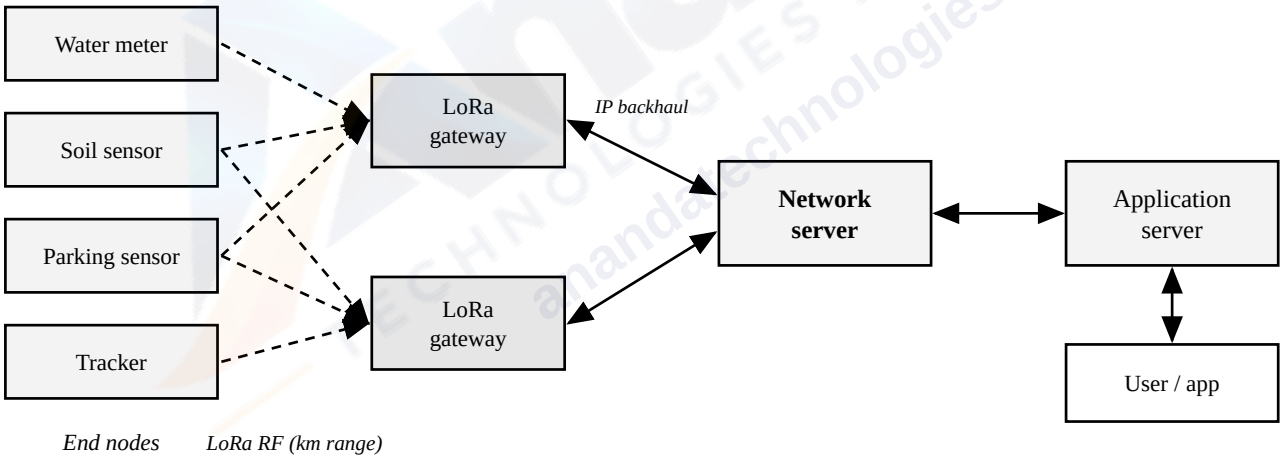


Fig.: LoRaWAN network architecture (star-of-stars)

- **LoRa** (Long Range) is a low-power wide-area (LPWAN) radio technology by Semtech using **chirp spread spectrum** modulation in licence-free sub-GHz ISM bands (865–867 MHz in India). Range 2–5 km in cities, 10–15 km or more in open areas; data rate 0.3–50 kbps; battery life of years.
- **End nodes** — battery-powered sensors that send small packets by LoRa radio.
- **Gateways** — receive packets from all nodes in range and forward them to the network server over IP (Ethernet / 4G). One node may be heard by several gateways (star-of-stars topology).

- **Network server** — removes duplicate packets, checks security, manages data rate (ADR). **Application server** — decrypts and uses the data, sends it to the user application.
- LoRaWAN is the MAC protocol defined by the LoRa Alliance; end-to-end AES-128 encryption; device classes A, B, C.

Applications (any two): smart water/gas metering, smart agriculture (soil sensors over large farms), asset and livestock tracking, smart parking, street-light monitoring.

(Diagram 2 M, explanation 1 M, two applications 1 M)

c) IoT design and security challenges. 4 M

Design challenges:

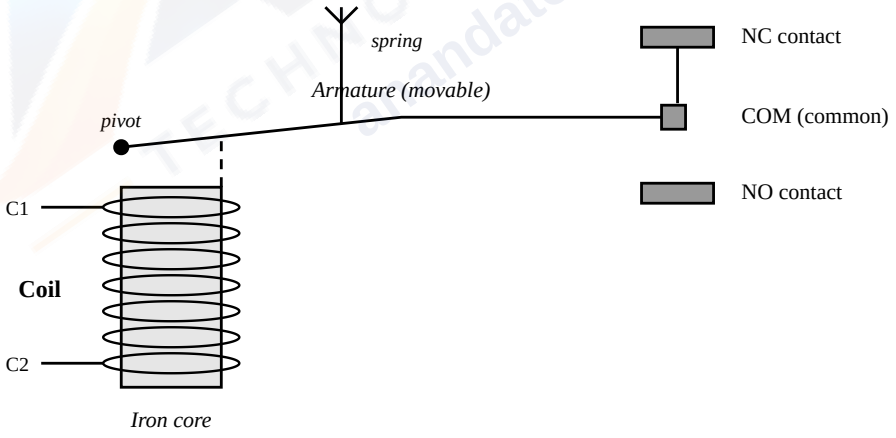
- **Interoperability** — devices of many vendors use different protocols and data formats.
- **Power / energy** — battery-operated nodes must work for years; needs low-power hardware and protocols.
- **Scalability** — the system must handle thousands of devices and huge data.
- **Reliability and latency** — continuous connectivity is not guaranteed; real-time applications need low delay.
- **Cost and size** of nodes.

Security challenges:

- **Weak authentication** — default/weak passwords (e.g. Mirai botnet attack).
- **Data privacy** — personal data (health, location) must be encrypted in transit and storage.
- **Insecure firmware / no updates** — devices are rarely patched.
- **Physical tampering** and attacks on the network (DoS/DDoS, man-in-the-middle).

(Any four challenges explained, 1 M each)

d) Relay – construction and working. 4 M



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

Fig.: Construction of an electromechanical relay (SPDT)

Construction: an electromechanical relay has (1) an **electromagnet** — coil wound on a soft-iron core, (2) a movable **armature** pivoted at one end, (3) a **spring** that holds the armature in the rest

position, and (4) **contacts** — COM (common, on the armature), NC (normally closed) and NO (normally open).

Working: when no current flows in the coil, the spring keeps COM touching NC. When the coil is energised (e.g. 5 V through a transistor driven by NodeMCU), the core becomes a magnet, attracts the armature, and COM moves from NC to **NO** — the load circuit (lamp, fan, pump on 230 V AC) connected through COM–NO is switched ON. When the coil current stops, the spring returns the armature. Thus a small DC signal controls a large AC load with **electrical isolation**. A flyback diode across the coil protects the driver transistor.

(Diagram 2 M, construction and working 2 M)

e) IoT cloud platform and ThingSpeak. 4 M

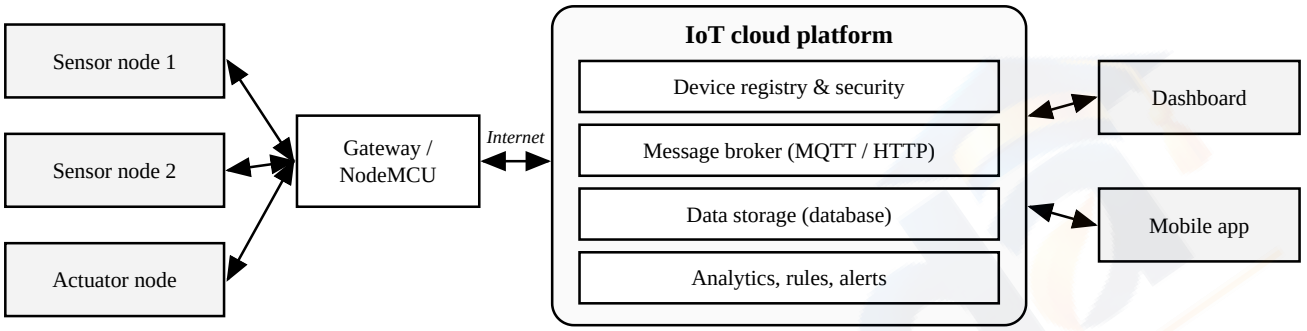


Fig.: Role of an IoT cloud platform

Role of an IoT cloud platform:

- Connects and manages devices securely (registration, authentication, keys).
- Receives data using HTTP-REST or MQTT and **stores** it in databases.
- **Analyses and visualises** data — charts, dashboards, rules, alerts.
- Sends **commands back** to actuators; provides APIs for mobile/web apps; scales to millions of devices.

ThingSpeak (by MathWorks): an open IoT analytics platform. Data is stored in **channels**; each channel has up to **8 fields** (e.g. field1 = temperature, field2 = humidity). A NodeMCU sends data with an HTTP GET/POST request using the channel's **Write API key** (or MQTT), e.g. `api.thingspeak.com/update?api_key=XXXX&field1=29.4`. ThingSpeak plots live charts, allows MATLAB analysis and actions (ThingTweet, React). The free licence allows one update about every 15 s.

(Role 2 M, ThingSpeak 2 M)

Q.5

a) IoT based greenhouse control system. 6 M

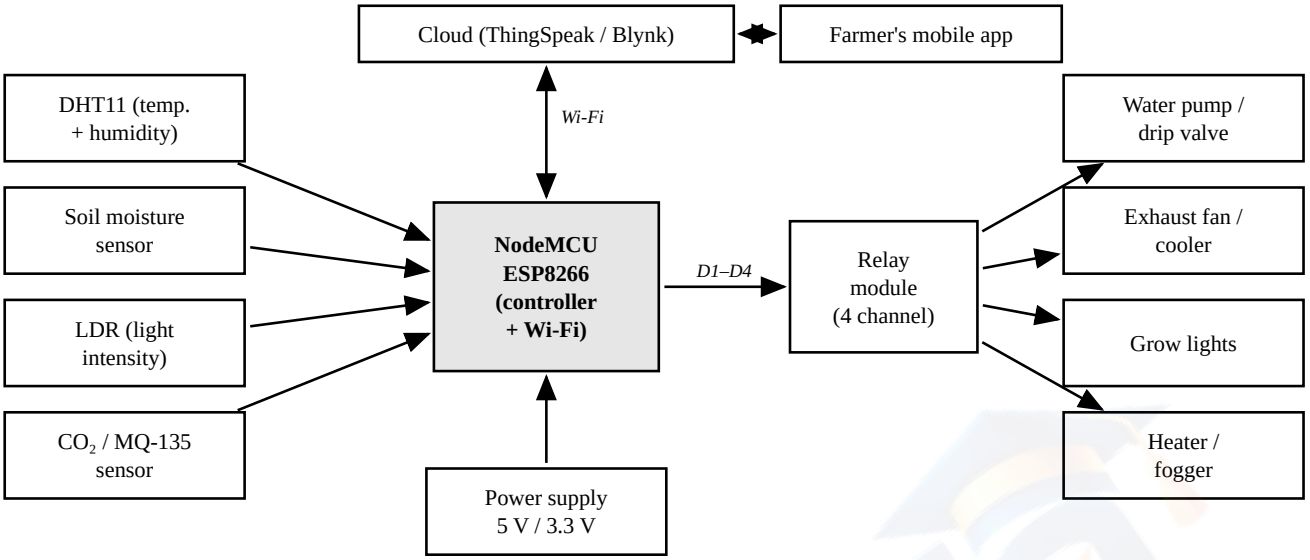


Fig.: Block diagram of IoT based greenhouse control system

A greenhouse must keep temperature, humidity, soil moisture and light at the best level for crops. IoT automates this and lets the farmer monitor it from a phone.

1. **DHT11** — measures air temperature and relative humidity.
2. **Soil moisture sensor** — measures water content of soil (analog/digital output).
3. **LDR** — measures light intensity for grow-light control.
4. **CO_2 / gas sensor (MQ-135)** — measures air quality inside the greenhouse.
5. **NodeMCU ESP8266** — reads all sensors, compares with set points and controls actuators; sends data to the cloud through built-in Wi-Fi.
6. **Relay module + actuators** — water pump/drip valve (soil dry), exhaust fan/cooler (temperature high), grow lights (low light), heater/fogger (too cold / humidity low).
7. **Cloud platform** (ThingSpeak/Blynk) — stores and plots data, sends alerts; **mobile app** — remote monitoring and manual override.
8. **Power supply** — 5 V/3.3 V for the controller and sensors.

Example logic: if soil moisture < 40 % → pump ON; if temperature > 32 °C → fan ON; if light low during day → grow lights ON.

(Block diagram 3 M, explanation 3 M)

b) DHT11 data format and decoding. 6 M

DHT11 uses a **single-wire (one-wire) serial** protocol on its DATA pin (with a 10 kΩ pull-up).

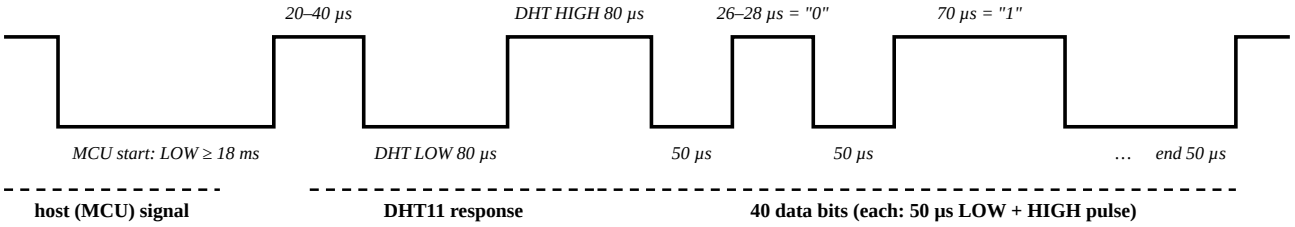


Fig.: DHT11 single-wire communication – timing diagram

1. **Start signal:** the MCU pulls DATA LOW for at least 18 ms, then releases it HIGH for 20–40 µs.
2. **Response:** DHT11 pulls the line LOW for 80 µs and then HIGH for 80 µs.
3. **Data:** DHT11 sends **40 bits**, MSB first. Every bit starts with a 50 µs LOW; the following HIGH lasts **26–28 µs for "0"** and **70 µs for "1"**.
4. After the last bit the sensor pulls the line LOW for 50 µs and releases it.

40-bit frame = 8-bit humidity integer + 8-bit humidity decimal + 8-bit temperature integer + 8-bit temperature decimal + 8-bit checksum. Checksum = last 8 bits of the sum of the first four bytes.

Humidity integer	*Humidity decimal*	*Temp. integer*	*Temp. decimal*	*Checksum*
00111110 = 62	**00000000** = 0	**00011101** = 29	**00000100** = 4	**01011111** = 95
bits 39 … 32	bits 31 … 24	bits 23 … 16	bits 15 … 8	bits 7 … 0

Fig.: Received 40-bit frame

Given: 0x3E, 0x00, 0x1D, 0x04, 0x5F → bit stream `00111110 00000000 00011101 00000100 01011111`

- Humidity = 62 + 0.0 = **62 % RH**
- Temperature = 29 + 0.4 = **29.4 °C**
- Checksum check: 62 + 0 + 29 + 4 = 95 = 0x5F; received checksum = 0x5F → equal, so **data is received correctly**.

(Timing diagram 2 M, frame format 2 M, calculation 2 M)

c) Technical specifications and pin diagram of NodeMCU ESP8266. 6 M

Specification	Value
Microcontroller / SoC	ESP8266EX (ESP-12E module), Tensilica L106 32-bit RISC
Clock speed	80 MHz (can run at 160 MHz)
Flash memory	4 MB (external SPI flash)
RAM	≈ 160 KB total (64 KB instruction RAM + 96 KB data RAM)
Operating voltage	3.3 V (logic level 3.3 V)
Input voltage	5 V through micro-USB or Vin pin (on-board AMS1117 3.3 V regulator)
Digital I/O pins	16 GPIO (D0–D8, RX, TX, SD2, SD3…); PWM on GPIOs (not D0)
Analog input	1 (A0), 10-bit ADC

Communication	UART (2, UART1 TX only), SPI, I2C (software), I2S
Wi-Fi	IEEE 802.11 b/g/n, 2.4 GHz; STA / AP / STA+AP modes; WPA/WPA2
USB-to-serial	CP2102 (or CH340); programmed from Arduino IDE / Lua / MicroPython
Current per GPIO	≈ 12 mA

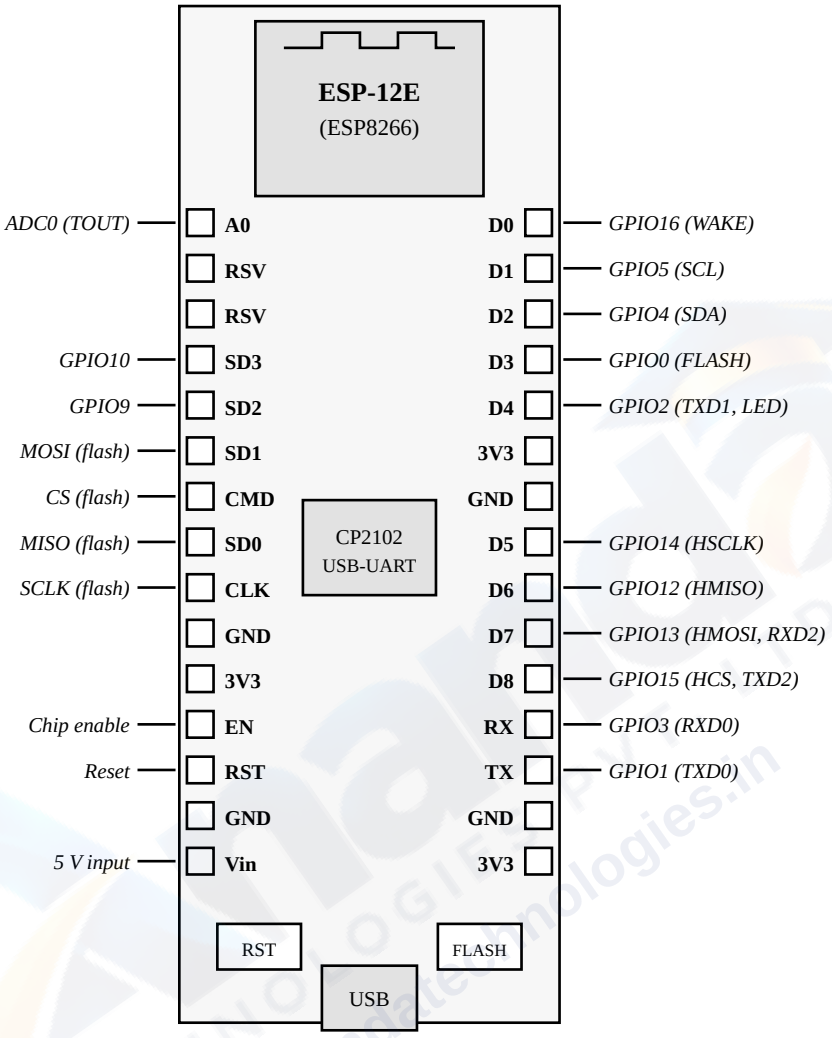


Fig.: Pin diagram of NodeMCU ESP8266 (ESP-12E, v1.0)

(Any six specifications 3 M, pin diagram 3 M)

Q.6

a) EV battery management system using IoT. 6 M

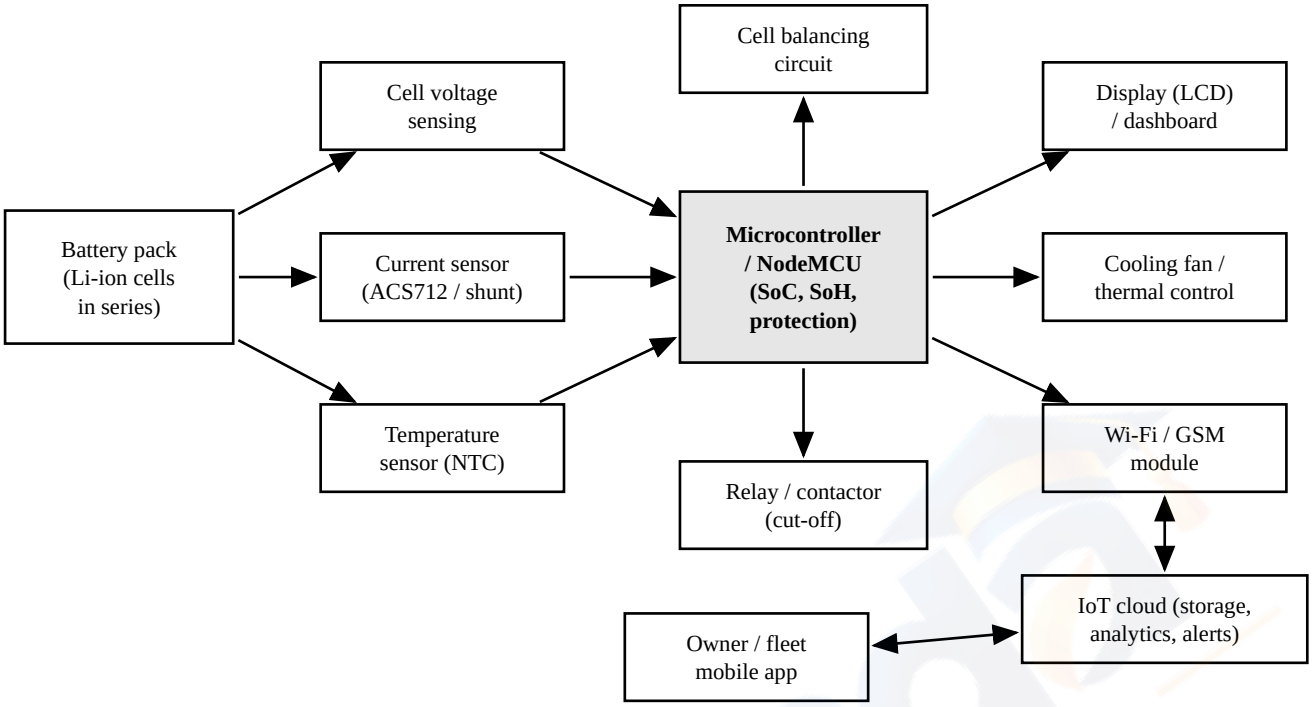


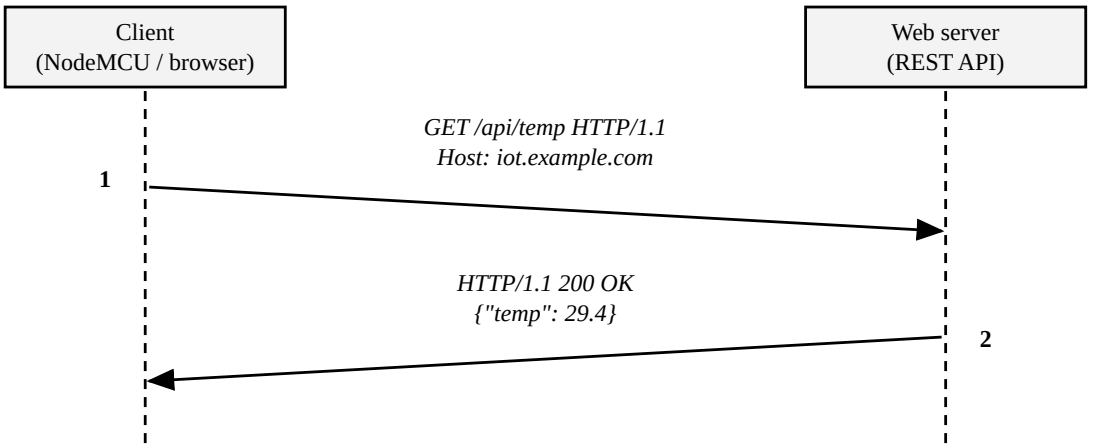
Fig.: Block diagram of EV battery management system (BMS) using IoT

A **battery management system (BMS)** monitors and protects the Li-ion battery pack of an electric vehicle; IoT adds remote monitoring and analytics.

1. **Battery pack** — many Li-ion cells in series/parallel that supply the motor.
2. **Voltage sensing** — measures each cell voltage (detects over-charge / deep discharge).
3. **Current sensor** (ACS712 / shunt) — charge and discharge current; used for coulomb counting of **SoC**.
4. **Temperature sensor** (NTC/LM35) — detects overheating of cells.
5. **Microcontroller / NodeMCU** — calculates **State of Charge (SoC)** and **State of Health (SoH)**, compares with safe limits and takes protective action.
6. **Cell balancing circuit** — equalises cell voltages (passive bleeding / active transfer). **Relay/contactor** — disconnects the pack on fault. **Cooling fan** — thermal management.
7. **Display** — shows SoC, range, alerts. **Wi-Fi/GSM module** — sends data to the **IoT cloud** for storage, analytics, fault prediction; the **mobile app** shows battery status, charging alerts and location to the owner/fleet manager.

(Block diagram 3 M, explanation 3 M)

b) HTTP request–response and methods. 6 M



Stateless: every request carries all information needed; TCP port 80 (443 for HTTPS)

Fig.: HTTP request–response communication

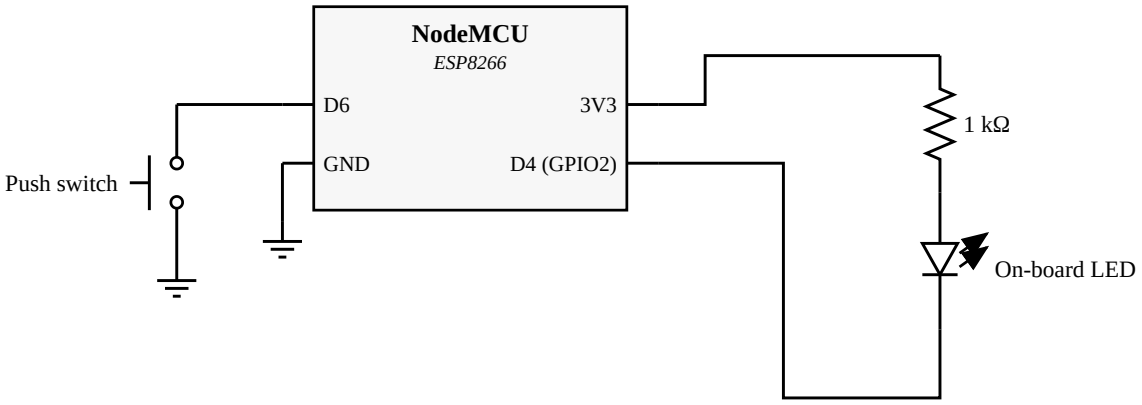
HTTP (HyperText Transfer Protocol) is an application-layer, **client–server, request–response** protocol that runs over TCP (port 80; HTTPS on 443). It is **stateless**. In IoT it is used in the **REST** style — every sensor/actuator is a *resource* identified by a URL.

1. The client (NodeMCU / browser / app) opens a TCP connection and sends a **request**: method + URL + headers (+ body).
2. The server processes it and returns a **response**: status code (200 OK, 201 Created, 404 Not Found, 500 Server Error) + headers + body (JSON/HTML).

Method	Meaning	IoT example
GET	Read a resource; no body; safe	`GET /api/devices/12/temp` → {"temp": 29.4}
POST	Create / send new data	`POST /api/readings` body {"temp": 29.4, "hum": 62}
PUT	Update / replace a resource	`PUT /api/devices/12/relay` body {"state": "ON"}
DELETE	Delete a resource	`DELETE /api/readings/501`

(Diagram 2 M, explanation 1 M, four methods 3 M)

c) Switch-controlled LED program; setup() and loop(). 6 M



Switch released → pin reads HIGH (internal pull-up); pressed → LOW. GPIO2 LOW → LED ON

Fig.: Push switch on D6 (INPUT_PULLUP) and on-board LED on GPIO2 (active LOW)

```
// Push switch on D6 controls the on-board LED (GPIO2 / D4, active LOW)
const int SW = D6;

void setup() {
  Serial.begin(9600);
  pinMode(SW, INPUT_PULLUP);        // released = HIGH, pressed = LOW
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  int state = digitalRead(SW);
  if (state == LOW) {               // switch pressed
    digitalWrite(LED_BUILTIN, LOW); // LOW turns the on-board LED ON
    Serial.println("Switch pressed  -> LED ON");
  } else {
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("Switch released -> LED OFF");
  }
  delay(200);
}
```

Output — Serial Monitor output (logic run on a PC with the switch released, pressed, pressed, released):

```
Switch released -> LED OFF
Switch pressed  -> LED ON
Switch pressed  -> LED ON
Switch released -> LED OFF
```

setup(): runs **only once** when the board is powered or reset. Used for initialisation — `Serial.begin()`, `pinMode()`, Wi-Fi connection, sensor begin.

loop(): runs **repeatedly forever** after setup(). Contains the main logic — read inputs, take decisions, drive outputs.

(Circuit 1 M, program 3 M, setup()/loop() 2 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