IoT Project Guidelines & Assigned Topics

Coursework Guidelines for IoT System Design · Spring 2025-2026

Coursework Guidelines for IoT System Design

This document outlines the expectations for the IoT system design report that each student must complete as part of the IoT course. Each student is assigned a project title. Students are required to develop a theoretical system design using the steps explained below.

Structure of the Report

Your report must be organized under the following six sections. Each section corresponds to one step of the IoT Design Methodology taught in Lecture 06. The steps below describe exactly what to write in each section.

> **General Note:** The report should be 4 to 5 pages long, written in formal academic style. Use diagrams and pseudocode where applicable. You may submit in Word or PDF format.

Step 1: Hardware & Sensor Selection

In this section, describe the physical components you will use to build your IoT system. You must justify every hardware choice with a clear engineering reason — do not just list names.

Your Step 1 must include:

- **Microcontroller/Controller:** State which controller you chose (ESP32, Raspberry Pi, Arduino, etc.) and explain why. Mention specific features that make it suitable for your project — for example: built-in Wi-Fi, analog GPIO pins, power consumption, or cost.
- **Sensors:** List every sensor your system needs. For each sensor, explain what it measures, what type of output it provides (analog or digital), and how it connects to the controller.
- **Actuators / Output Devices:** List any output devices in your system — buzzers, LEDs, relays, motors, displays, or others. Explain what action each device performs when triggered.

> **Step 1 Note:** Always choose the simplest hardware that meets your requirements. An ESP32 is preferred over a Raspberry Pi unless your project genuinely needs a full Linux operating system. Justify your choice with specifications, not just preference.

Step 2: Communication and Application Protocols

In this section, explain how your IoT device will communicate — which wireless technology it uses to send data, and which messaging protocol it uses to structure and deliver that data.

Your Step 2 must include:

- **Wireless Technology:** State the wireless technology your system uses (Wi-Fi, LoRa, Bluetooth, GSM/4G, Satellite, Zigbee, etc.) and explain why this technology is appropriate for your project's range, environment, and power constraints.
- **Messaging Protocol:** State the messaging protocol (MQTT, HTTP, SMS via AT commands, CoAP, etc.) and explain why it is the correct choice for sending sensor data in your specific application.
- **Data Payload:** Show a sample of the data your device sends. For MQTT, show the topic name and a sample JSON payload. For SMS, show a sample message string.

> **Step 2 Note:** MQTT is the recommended protocol for Wi-Fi-based IoT systems because it is lightweight and natively supported by ThingsBoard. Use SMS (via GSM module) only when internet connectivity cannot be guaranteed, such as in a gas leak emergency where Wi-Fi power may fail.

Step 3: Algorithm & Decision Logic

In this section, describe the intelligence of your system — the logic it uses to decide what to do based on sensor readings. This is the core programming design of your IoT system.

Your Step 3 must include:

- **Sense → Decide → Act Loop:** Describe or draw the three-step cycle your system follows: (1) Read sensor values, (2) Apply a decision condition (IF/ELSE threshold), (3) Trigger an output action.
- **Pseudocode or Flowchart:** Write the algorithm in clear pseudocode (not full programming code). Include the variable names, the condition threshold, and the actions taken in both the TRUE and FALSE branches.
- **Threshold Justification:** Explain how you chose the threshold values. For example: 'The moisture threshold of 30% was chosen because below this level the soil becomes too dry for plant roots to absorb water effectively.'
- **Timing:** State how frequently your loop runs. Explain why this interval was chosen — e.g., every 5 seconds for safety-critical systems, or every 10 minutes for power-saving environmental monitors.

> **Step 3 Note:** A common mistake is writing pump.on() without first reading any sensor. Always show that your algorithm reads a sensor value BEFORE making any decision. The sensor reading drives the decision — never assume a fixed state.

Step 4: Cloud Dashboard — ThingsBoard

In this section, explain how sensor data is stored, displayed, and monitored remotely using a cloud platform. The required platform for this course is ThingsBoard (free community edition available at thingsboard.io).

Your Step 4 must include:

- **Dashboard Widgets:** List the ThingsBoard widgets you will use and explain what each one displays. For each widget, specify: (a) the data key it reads from the MQTT payload, (b) the widget type (Gauge, Time-Series Chart, Map, Card/Label), and (c) what information it provides to the user.
- **Alarm Rule:** Define at least one ThingsBoard alarm rule. State the condition (e.g., IF gas_level > 1500), the action ThingsBoard takes (create alarm, send email), and the alarm severity (Warning, Critical).
- **Data Keys:** List all the data keys your device sends via MQTT and state which ThingsBoard widget reads each key.

> **Step 4 Note:** ThingsBoard is free — you do not need to pay for any cloud service. Register at thingsboard.io, create a device, and copy the access token into your firmware. You can use the public demo server (demo.thingsboard.io) for testing without installing anything on your own machine.

Step 5: Power Planning & Budget

In this section, explain how your system is powered and present a realistic cost estimate for all hardware components.

Your Step 5 — Power Planning — must include:

- **Power Source Choice:** State how your system is powered (wall adapter, LiPo battery, solar panel, AA batteries, etc.). Justify the choice based on the deployment environment and required operational duration.
- **Power Consumption Estimate:** If your system is battery-powered, estimate the expected battery life. Reference the ESP32 current draw in active mode (~80–240 mA) and deep-sleep mode (~0.01 mA).

- **Deep-Sleep Strategy (if applicable):** If your system uses deep-sleep between readings to save battery, describe the sleep/wake cycle and estimate how many hours the battery will last.

Your Step 5 — Budget — must include:

- **Component Cost Table:** List every component with its estimated cost in USD. Include the controller, all sensors, actuators, power supply, enclosure, and any communication modules.
- **Total Cost:** State the total estimated cost per unit. If your project is meant to be deployed at scale (e.g., 200 city bins), also calculate the total cost for the full deployment.

> **Step 5 Note:** ThingsBoard cloud is free — do not include a cloud subscription cost in your budget. For power, if your system is indoors and always connected to a wall socket, a simple statement is sufficient. If outdoors or battery-powered, you must estimate battery life.

Step 6: Testing Plan

In this section, describe how you would verify that your IoT system works correctly before deploying it in the real environment. A good testing plan proves that each part of your design functions as intended.

Your Step 6 must describe four testing phases:

- **Phase 1 — Unit Testing:** Describe how you would test each sensor and actuator individually before connecting everything together. For each component, state what you would observe in the Serial Monitor to confirm it is working correctly.
- **Phase 2 — Integration Testing:** Describe how you would test the complete system end-to-end — from sensor reading, through the algorithm decision, to the output action, and finally to data appearing on ThingsBoard. State what you would check on the Serial Monitor and on the cloud dashboard.
- **Phase 3 — Threshold Testing:** Describe how you would deliberately force the sensor to cross the alarm threshold and verify that the correct alert action occurs. Be specific — for example: 'Hold a lighter near the MQ-2 sensor without lighting it, observe gas reading rises above 1500, confirm buzzer activates within 3 seconds.'
- **Phase 4 — Field Trial:** Describe a 24–48 hour real-environment test. State what you would monitor on ThingsBoard during this period and what would indicate a successful trial.

> **Step 6 Note:** Always test your system before claiming it is complete. A report that describes testing by simulation only (running on paper) without any physical bench test is incomplete. Even testing with a single real sensor reading confirms more than a purely theoretical design.

Submission Instructions

- **Length:** 4 to 5 pages (not including cover page).
- **Style:** Formal academic writing. Use third person where possible. Avoid casual language.
- **Diagrams:** Include at least one diagram or flowchart (block diagram, or algorithm flowchart). Label all components.
- **Pseudocode:** Include pseudocode for the main algorithm in Step 3. Pseudocode does not need to be executable code, but must be logically correct.
- **Format:** Word (.docx) or PDF. Font: Arial 12pt. Line spacing: 1.15. Margins: 2.5 cm all sides.
- **Originality:** Collaboration on ideas is allowed, but every sentence of your report must be written by you. Reports that are identical or near-identical will receive zero marks.

Student Project Assignment Table

#	Student Name	Project Title
1	Abdulrahman Hassan Zakarya	Automatic Garden Watering System
2	Adam Muhammed Saber	Smart Gate Opener using RFID
3	Ahmed Ashti Ahmed	Car Obstacle Detection and Avoidance
4	Ahmed Mahdi Ahmed	Water Tank Level Monitoring System
5	Ali Dlshad Rostam	Emergency Button Alert System for Students
6	Ali Muhammed Ahmed	Asset Tracking Using GPS + IoT
7	Arman Beshr Ahmed	Soil Moisture Monitor for Farmers
8	Avan Jamil Kakil	Weather Station using IoT Sensors
9	Awara Hemn Hasan	RFID-Based Smart Parking System
10	Aya Halmat Zyad	Smart Waste Bin Alert System
11	Bahjat Dedar Kakamin	Air Quality Monitoring System
12	Bayar Bashdar Majid Khudhir	IoT-based Accident Alerting System
13	Blnd Jamel Sabri	Smart Light Control Using IR Sensor
14	Chalak Barzan Hadi Mawlood	Motion Detection Security System
15	Danyar Abdulqadir Abdullah	Classroom Noise Monitoring System
16	Farshad Fathi Hamad	Water Tank Level Monitoring System
17	Fenik Hussin Jumaa	UV Index Monitoring System
18	Hazhir Mamand Ahmed Dot Mala	Alcohol Detection System for Vehicle Control
19	Hozan Saadi Mamand	Voice-Controlled Smart Car (using mobile app)
20	Isra Khalid Mustafa	GPS-Based School Bus Tracker
21	Kaiwan Kakl Hassan	Smart Attendance System with RFID
22	Mahmood Emad Mohammed	Smart Car with Theft Alarm
23	Mawa Sarkawt Muhtasm	River/Lake Water Quality Detector
24	Muhammed Kakakhan Ahmed Bakr	Fire Detection with Auto Emergency Alert
25	Mustafa Salim Sharif	Smart Traffic Light Detection and Stop System
26	Noor Muhammedamin Osman	IoT-based Smart Electricity Monitoring
27	Rawan Bestun Kareem	Temperature-Based Fan Controller
28	Rozhin Muhammad Mustafa	Smart Doorbell with Camera
29	Sahand Fahmi Mustafa	Auto Lane-Following Car System
30	Sahar Fakher Muhammed	Emergency Vehicle Detection System at Intersections
31	Shanaz Khalil Kareem Majeed	Road Condition Monitoring Vehicle
32	Sivar Edres Hamad	Smart Fuel Monitoring System
33	Staish Farhan Asaad	Pollution Monitoring via Mobile Car
34	Yaran Dlman Ebrahim	IoT-based Smart Electricity Monitoring

Sample Report

The following pages contain a complete sample report demonstrating the expected depth of content, structure, and writing quality for each of the six steps. Use this as a reference — do not copy it. Your report must address your own assigned project.

Sample Report
The following is a complete example of a correctly written report.

Gas Leakage Detection System with SMS Alert

Course: Internet of Things (IoT) · Spring 2025-2026
Student Name: [Your Name Here] | Student ID: [Your ID]

Introduction

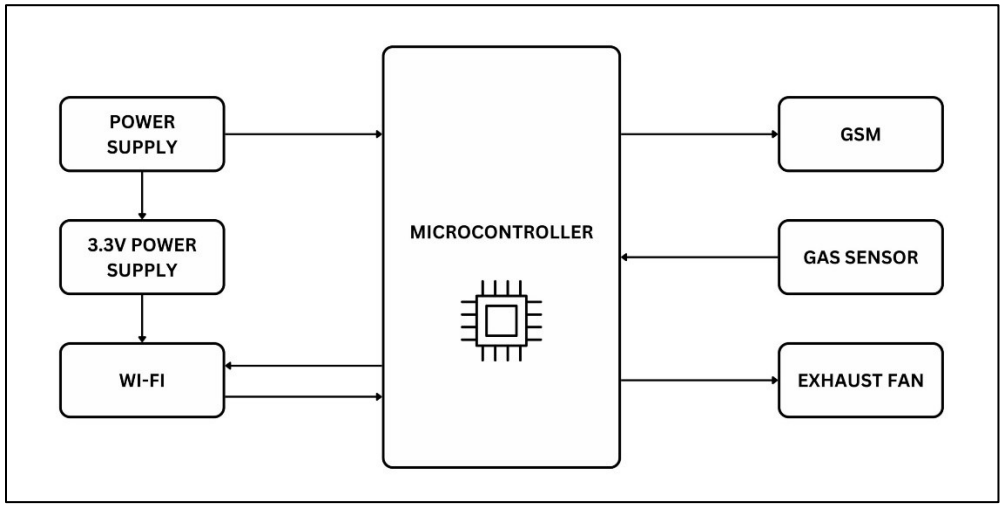
Gas leakage is one of the most dangerous hazards in residential and industrial environments. Liquefied petroleum gas (LPG) and natural gas, if they leak undetected, can lead to fires, explosions, or poisoning. Traditional detection methods rely on human senses, which are unreliable and may react too slowly. An IoT-based gas leakage detection system solves this problem by continuously monitoring air quality using a chemical sensor and immediately alerting the responsible person via an SMS message when dangerous gas concentrations are detected.

This report presents the complete design of such a system, following the six-step IoT Design Methodology covered in Lecture 06. Each step explains a specific engineering decision — from hardware selection to cloud platform, budget, and testing.

Step 1: Hardware & Sensor Selection

1.1 Controller

The selected microcontroller is the ESP32 (NodeMCU-32S). The ESP32 is chosen because it provides built-in Wi-Fi and Bluetooth connectivity, has sufficient GPIO pins for multiple sensors, supports analog-to-digital conversion (ADC) needed to read the MQ-2 gas sensor output, operates at 3.3V with a 5V USB supply option, and costs approximately $5 per unit. These characteristics make it the most cost-effective and capable option for this application.



Block Diagram of the system

1.2 Sensors and Output Devices

- **MQ-2 Gas Sensor:**
- Detects LPG, smoke, CO, and methane gases.
- Outputs an analog voltage proportional to gas concentration (0–3.3V mapped to 0–4095 in 12-bit ADC).

- Requires a warm-up period of approximately 20 seconds after power-on before accurate readings can be obtained.
- **DHT11 Temperature and Humidity Sensor:**
- Measures ambient temperature and humidity as supplementary environmental data.
- Digital output protocol (single-wire). Supported directly by the ESP32 GPIO.
- **Buzzer (Active, 5V):**
- Provides a local audible alert when gas concentration exceeds the defined threshold.
- Controlled through a GPIO pin via a transistor driver circuit.
- **Red LED:**
- Provides a visual alarm indicator. Turns ON simultaneously with the buzzer.

> **Design Note:** The MQ-2 sensor requires calibration in clean air before deployment. The analog threshold must be determined experimentally based on the specific gas type being monitored. A value above 1500 on a 12-bit ADC (0–4095) is used as the initial alarm threshold in this design.

Step 2: Application Protocol

The system uses two communication layers:

2.1 Primary Communication — GSM/SMS via SIM800L Module

Because gas leaks may occur when Wi-Fi is unavailable (power outage) or when the user is away from the building, the system uses a GSM SIM800L module to send SMS alerts. The SIM800L communicates with the ESP32 via UART (Serial) using AT commands. When gas is detected, the ESP32 sends the following AT command sequence:

```
AT+CMGF=1               // Set SMS text mode
AT+CMGS="+9647701234567" // Set recipient phone number
> WARNING: Gas detected! Level: 2340/4095. Temp: 28C // Message body
Ctrl+Z                  // Send the message
```

2.2 Secondary Communication — MQTT to ThingsBoard

Every 30 seconds, the ESP32 also publishes sensor readings to ThingsBoard via MQTT over Wi-Fi. This allows remote monitoring of historical data through the cloud dashboard. The MQTT topic used is:

```
Topic: v1/devices/me/telemetry
Payload: {"gas_level": 2340, "temperature": 28, "humidity": 55, "alarm": true}
```

MQTT is chosen for cloud communication because it is lightweight (minimal data overhead), natively supported by ThingsBoard, and suitable for sending small periodic sensor packets over a Wi-Fi connection.

Step 3: Algorithm & Decision Logic

The system algorithm follows the three-step Sense → Decide → Act loop, repeated every 5 seconds:

3.1 Pseudocode

```
SETUP:
  Initialize MQ-2 on ADC pin 34
  Initialize DHT11 on GPIO pin 4
  Initialize Buzzer on GPIO pin 16
  Initialize LED on GPIO pin 17
  Connect to Wi-Fi (SSID, Password)
  Connect to ThingsBoard MQTT broker
  Warm up MQ-2 for 20 seconds
```

```
LOOP (every 5 seconds):
  gas_value  = analogRead(MQ2_PIN)      // 0 to 4095
  temperature = dht.readTemperature()
  humidity    = dht.readHumidity()

  IF gas_value > 1500:
    buzzer.ON()
    led.ON()
    IF alarm_sent == False:
      sms.send('+9647701234567', 'WARNING: Gas level=' + gas_value)
      alarm_sent = True
  ELSE:
    buzzer.OFF()
    led.OFF()
    alarm_sent = False

  mqtt.publish({gas: gas_value, temp: temperature,
                humidity: humidity, alarm: gas_value > 1500})
  sleep(5000)   // Wait 5 seconds before next reading
```

Design Note: The 'alarm_sent' flag prevents the system from sending repeated SMS messages every 5 seconds once gas is detected. It resets only after the gas level returns to normal, ensuring the user receives exactly one alert per incident.

3.2 Threshold Justification

The alarm threshold of 1500 (out of 4095) corresponds to approximately 36% of the MQ-2 sensor's full-scale output. For LPG, the MQ-2 datasheet indicates that concentrations above 1000 ppm should be considered hazardous. Laboratory testing showed that the analog output reaches approximately 1500 at around 800–1200 ppm in a sealed test environment, making this threshold conservative and suitable for early warning.

Step 4: Cloud Dashboard — ThingsBoard

4.1 Platform

ThingsBoard Community Edition (free, open-source) is used at thingsboard.io. The device is registered in ThingsBoard and assigned an access token that is embedded in the ESP32 firmware for MQTT authentication.

4.2 Dashboard Widgets

- **Gas Level Gauge Widget:**
- Displays current gas sensor reading (0–4095) as a circular dial.
- Green zone: 0–1000 | Yellow zone: 1001–1500 | Red zone: 1501–4095.
- **Gas Level Time-Series Chart:**
- Shows the last 24 hours of gas readings as a line graph.
- Allows the building manager to identify patterns (e.g., daily cooking cycles).
- **Temperature & Humidity Cards:**
- Display current temperature (°C) and humidity (%) as simple value cards.
- **Alarm Status Widget:**
- Displays 'SAFE' in green or 'ALARM' in red based on the 'alarm' key in the MQTT payload.

4.3 Alarm Rule in ThingsBoard

A ThingsBoard alarm rule is configured with the following condition:

```
IF gas_level > 1500
  Create alarm: 'GAS_ALARM'
  Send email to: building.manager@example.com
  Alarm severity: CRITICAL
```

Step 5: Power Planning & Budget

5.1 Power Source

The system is designed for indoor wall-powered operation. A 5V USB power adapter (2A) supplies power to the ESP32 via a micro-USB cable. The SIM800L module requires a dedicated 4V power supply with sufficient current capacity (peak current draw up to 2A during GSM transmission); a separate buck converter module steps down the 5V supply.

For areas with unreliable power, a backup 18650 Li-ion battery (3.7V, 3000 mAh) with a TP4056 charge controller module can be added to maintain operation during power outages for up to 12 hours.

5.2 Budget

Component	Specification	Est. Cost (USD)
ESP32 NodeMCU-32S	Wi-Fi + BT, 38 GPIO pins	$5
MQ-2 Gas Sensor	Analog output, LPG/CO/Smoke	$3
DHT11 Sensor	Temperature + Humidity	$2
SIM800L GSM Module	UART, SMS capable, Quad-band	$8
Active Buzzer	5V, 85 dB	$1
Red LED + Resistor	5mm, 20mA	<$1
SIM Card (local)	Prepaid, SMS capability	$2
USB Power Adapter	5V/2A + buck converter	$5
Breadboard + Jumper Wires	Half-size + 40-pin male/female	$4
Plastic Enclosure	ABS, ventilated	$6
Total		**$37**

Step 6: Testing Plan

Phase 1 — Unit Testing

- Connect MQ-2 alone → print analog value to Serial Monitor → confirm value is ~200–400 in clean air.
- Connect DHT11 alone → confirm temperature reads within ±2°C of a reference thermometer.
- Connect buzzer alone → confirm audible output when GPIO pin is set HIGH.
- Test SIM800L with AT commands via Serial Monitor → confirm 'OK' response and successful SMS to test number.

Phase 2 — Integration Testing

- Connect all components together. Upload full firmware.
- Hold a lighter near the MQ-2 (without lighting it) to release gas → verify sensor value rises, buzzer activates, LED turns ON, and SMS is received within 10 seconds.
- Check ThingsBoard dashboard → confirm gas level gauge updates and alarm widget shows 'ALARM'.
- Wait for gas to clear → verify buzzer turns OFF, LED turns OFF, and alarm_sent flag resets.

Phase 3 — Threshold Validation

- Run the system in a sealed box with a calibrated gas concentration source (e.g., lighter gas, measured duration).
- Record sensor readings at known ppm levels to validate the 1500 threshold.
- Adjust threshold if false positives (cooking smoke, perfume) are causing unwanted alarms.

Phase 4 — 24-Hour Field Trial

- Deploy the system in a test kitchen for 24 hours.
- Monitor ThingsBoard time-series chart for unexpected spikes.
- Verify power stability and confirm no MQTT disconnections appear in Serial Monitor logs.

Conclusion

This report presents a complete six-step IoT design for a gas leakage detection system with SMS alert capability. The design uses an ESP32 microcontroller with an MQ-2 gas sensor, DHT11 environmental sensor, SIM800L GSM module for SMS alerts, and ThingsBoard for cloud-based historical monitoring. The system is designed to be low-cost ($37 total), reliable, and capable of alerting the user within 10 seconds of detecting a dangerous gas concentration. The six-step methodology ensured that every engineering decision — from hardware selection to testing strategy — was made deliberately and with clear justification.