Name: Saad Saddique
Reg no: 22-NTU-CS-1370

Assignment: Cloud and Gateway implementation

SHORT QUESTION:

1. **What is ThingSpeak and why is it used in IoT applications?**
   ThingSpeak is an open-source IoT analytics platform that enables real-time data collection, visualization, and analysis from sensors and devices over the Internet.
2. **How does MQTT differ from HTTP in terms of communication for IoT devices?**
   MQTT is a lightweight, publish-subscribe protocol optimized for low-bandwidth, low-power devices, unlike HTTP which is heavier and request-response based.
3. **Why is multi-core tasking important in an ESP32-based IoT system?**
   ESP32 has dual cores, allowing simultaneous tasks—like sensor reading and Wi-Fi communication—without blocking or delays.
4. **What is a mutex/lock, and why is it needed when updating an OLED display concurrently?**
   A mutex is a mutual exclusion object that prevents multiple threads from accessing shared resources like OLED displays at the same time, ensuring data consistency.
5. **Name two features of Firebase that make it suitable for real-time sensor data logging.**
   - Real-time Database Sync
   - Built-in Authentication and Security Rules
6. **What is InfluxDB, and why is it preferred over traditional databases for IoT data?**
   InfluxDB is a time-series database optimized for high-write loads and time-stamped data, making it ideal for sensor-based logging and analytics.
7. **Define time-series data and give one example from this assignment.**
   Time-series data is a sequence of data points indexed by time; e.g., DHT22 temperature readings every minute.
8. **What is the purpose of NTP synchronization in IoT systems?**
   NTP ensures accurate timekeeping on IoT devices, which is critical for time-stamped data logging and analysis.

MEDIUM LENGTH QUESTIONS:

9. ESP32 to ThingSpeak Flow Design

Steps:

1. **Wi-Fi Connection**

```
import network
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('SSID', 'PASSWORD')
```

2. **Sensor Reading (DHT22)**

```
import dht, machine

sensor = dht.DHT22(machine.Pin(4))

sensor.measure()

temp = sensor.temperature()

hum = sensor.humidity()
```

3. **Prepare HTTP GET Request with API Key**

```
import urequests
api_key = "YOUR_API_KEY"
url =
f"https://api.thingspeak.com/update?api_key={api_key}&field1={temp}&field2={hu
m}"
response = urequests.get(url)
```

Diagram (simplified):

[DHT22 Sensor] → [ESP32] → [Wi-Fi] → [ThingSpeak Cloud]

| |

Readings HTTP GET with API Key

10. Arduino-C vs MicroPython for IoT Development:

Feature	Arduino (C++)	MicroPython
Ease of Use	Complex syntax, steep learning curve	Pythonic, beginner-friendly
Performance	High performance, close to hardware	Slightly slower due to interpretation
Hardware Ctrl	Full control over peripherals	Limited support on some advanced features
Ecosystem	Mature community and library base	Growing, but fewer libraries
Debugging	Limited real-time debugging	Easy with REPL and logs

Conclusion:
Arduino is better for performance-critical applications, while MicroPython suits rapid prototyping and educational use.

11. Firebase Data Flow Design

Steps:

1. **NTP Sync for Accurate Timestamps**

```
import ntptime

ntptime.settime()
```

2. **Prepare Data in JSON Format**

```
import ujson
data = ujson.dumps({
  "timestamp": time.time(),
  "temperature": temp,
  "humidity": hum
})
```

3. **Send Data to Firebase via REST API**

   headers = {"Content-Type": "application/json"}

   urequests.post(FIREBASE_URL, headers=headers, data=data)

4. **Authentication (Optional)**

   Use Firebase Auth token in headers for secure access.

12. Environmental Classification Using Python & InfluxDB

Workflow:

1. **Fetch Data from InfluxDB**

   from influxdb import InfluxDBClient

   client = InfluxDBClient(host='localhost', port=8086, database='iot')

   result = client.query('SELECT * FROM sensor_data')

2. **Preprocess Data**
   - Handle missing values
   - Normalize temperature and humidity

3. **Train Model**

   from sklearn.tree import DecisionTreeClassifier

   clf = DecisionTreeClassifier()

   clf.fit(X_train, y_train)

4. **Classify and Store Back**

   prediction = clf.predict([[28, 60]]) # example input

   json_body = [{

   "measurement": "classification",

   "fields": {"status": prediction[0]}

   }]

client.write_points(json_body)

Long Answer / Real Design Question

13(a). Gateway vs Cloud-Based Architecture

Feature	Gateway (Edge Computing)	Cloud-Based
Deployment	Local processing hub (e.g., Raspberry Pi)	Cloud platforms like Firebase or AWS
Latency	Low (processed near the source)	Higher (network-dependent)
Processing	Done locally, efficient for real-time tasks	Done in cloud, good for heavy analytics
Scalability	Limited by local resources	Highly scalable with cloud infrastructure

13(b). Layer Comparison

Layer	Gateway-Based	Cloud-Based
Sensing	DHT22 sensor on ESP32	Same
Communication	MQTT/Wi-Fi to local gateway	MQTT/HTTP to cloud
Storage	SQLite/InfluxDB locally	Firebase, AWS, InfluxDB cloud
Visualization	Local dashboard (Grafana on Pi)	ThingSpeak, Cloud Grafana
Decision-Making	On-device ML model	AI model in cloud

1. **Gateway-Based**

[Sensors] → [ESP32] → [MQTT] → [Gateway Device] → [Local InfluxDB + Grafana + AI]

2. **Cloud-Based**

[Sensors] → [ESP32] → [Wi-Fi/HTTP] → [Firebase/ThingSpeak] → [Cloud Dashboard + AI]

Bonus Question: AI-Based Environmental Prediction Using Edge Computing

Edge AI models on ESP32 can analyze historical data (e.g., temperature, humidity) and predict future states like "Hot" or "Comfortable." This prediction:

- **Improves responsiveness** by eliminating reliance on cloud latency.
- **Reduces bandwidth** since only decisions or alerts are sent, not raw data.
- **Trade-offs:** Limited model size and computational resources on ESP32; suitable for simple ML models like decision trees or regression.
- **Implementation:** Use offline-trained models and convert them to formats (e.g., TinyML, uTensor).
- **Applications:**
    - Smart agriculture (predict soil moisture needs)
    - HVAC systems (auto-adjust room climate)
    - Wearables (alert based on environmental condition)
- **Example:** If humidity > 70% and temp > 30°C → Predict "Humid-Hot" and activate fan via GPIO pin.