Smart Chair Occupancy Detection using IoT – Project Overview

🔍 Objective

To develop an IoT-based system that detects whether a chair is *occupied* or *unoccupied* using a Force Sensitive Resistor (FSR) and transmits this status along with GPS location to an online database (or optionally to a mobile app). This helps in locating free chairs in public or institutional settings.

⚙️ Key IoT Components Used

1. **ESP32 Dev Module**
   - Microcontroller with WiFi & Bluetooth support
   - Acts as the brain of the system, reads FSR sensor and GPS data, and sends them over WiFi.
2. **FSR (Force Sensitive Resistor)**
   - Used to detect pressure/weight (presence of a person).
   - Changes its resistance based on the applied force.
3. **NEO-6M GPS Module**
   - Provides real-time GPS coordinates of the chair.
   - Communicates with ESP32 via serial communication.
4. **Power Source**
   - A USB power bank (or direct USB from laptop) powers the ESP32.

🔗 Connection Summary

- **FSR**:
  - One leg to 3.3V on ESP32
  - Other leg of FSR → Connect to:
  - Analog pin D32 (or any other ADC-capable GPIO like D33, D34)
  - 10kΩ resistor connected between this leg and **GND** (acts as a voltage divider)

□ **Why Voltage Divider?**
The FSR by itself doesn't produce a voltage. The resistor helps create a voltage that the ESP32 can measure, which changes with applied force.

- **GPS (NEO-6M)**:
  - VCC to VIN of ESP32 (if 5V GPS module)
  - GND to GND
  - TX to RX2 (GPIO16)

   - RX to TX2 (GPIO17)

□ Working Principle

1. **Chair Monitoring Logic**:
   - When someone sits on the chair, the FSR detects pressure.
   - The ESP32 reads the analog value from the FSR and compares it with a **threshold** to determine if the chair is occupied.
2. **GPS Tracking**:
   - NEO-6M GPS module provides real-time latitude and longitude data.
   - This helps in uniquely identifying and locating each chair.
3. **Data Handling**:
   - Based on FSR and GPS input, ESP32 prepares a data packet (e.g., chair ID, status, coordinates).
   - This data can optionally be sent over WiFi to cloud platforms like Firebase, Ubidots, or displayed via Serial Monitor for local testing.

IoT Code Flow Summary (Arduino IDE)

1. **Initialize FSR & GPS pins**
2. **Read analog data from FSR**
3. **Compare FSR value with a threshold**
4. **If value > threshold → Occupied**
5. **Read GPS coordinates using TinyGPS++**
6. **Print/send data (chair status + location)**

Threshold Logic

- FSR output value ranges from 0 to 4095 (for ESP32)
- A mid-range threshold is set (e.g., 1000)
  - `if (fsrValue > 1000) → chair is OCCUPIED`
  - else → chair is FREE

Communication Protocols

- **Serial Communication**:
  - Between ESP32 and GPS module (using UART pins)
- **WiFi (Optional)**:
  - ESP32 can send real-time data to the cloud if integrated.

Deployment/Real-World Use

- Each chair has its own ESP32+FSR+GPS setup
- Chairs can be placed in public spaces like libraries, halls, campuses
- Monitoring app/website shows:
    - Chair location
    - Occupancy status
    - Distance from user (optional)

Advantages

- Real-time visibility of seat availability
- Location-aware seat finding
- Efficient use of resources (space)
- Easy to scale with more chairs

Code BreakDown

1. Libraries Included

```
#include <HardwareSerial.h>
#include <TinyGPSPlus.h>
```

- `HardwareSerial.h`: Allows using additional UART (Serial) ports on the ESP32 (like Serial1, Serial2).
- `TinyGPSPlus.h`: A library to parse and interpret GPS data from NEO-6M.

📡 2. GPS Setup

```
TinyGPSPlus gps;
HardwareSerial gpsSerial(1); // Using Serial1
```

- `gps`: Object for decoding GPS data.
- `gpsSerial`: Sets up **Serial1**, which communicates with the **GPS module** via GPIO16 (RX2) and GPIO17 (TX2).

□ 3. FSR Sensor Setup

```
const int fsrPin = 34;         // Analog pin
const int threshold=1800;// Threshold value for detecting seated weight
(~35kg)
```

- `fsrPin`: Connected to the output of your **FSR voltage divider**.
- `threshold`: The analog value above which you assume **someone is seated**.

⚙□ 4. setup() Function

```
void setup() {
  Serial.begin(115200); // Serial monitor for debugging
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17); // GPS comm: baud rate 9600,
RX=16, TX=17

  Serial.println("System starting...");
}
```

- Initializes Serial Monitor and GPS Serial communication.
- Prints a startup message.

🔁 5. loop() Function – Main Execution Logic

□ (a) FSR Value Reading

```
int fsrValue = analogRead(fsrPin);
```

- Reads the pressure applied to the FSR.
- Analog values range from 0 (no pressure) to ~4095 (maximum pressure).

□ (b) Check Seated or Not

```
if (fsrValue > threshold) {
  Serial.println("Weight Detected: Person is seated!");
} else {
  Serial.println("No one is seated.");
}
```

- If analog value is greater than 1800, it assumes someone is sitting.

🛰□ (c) GPS Reading

```
while (gpsSerial.available() > 0) {
  gps.encode(gpsSerial.read());
}
```

- Continuously reads available characters from GPS and feeds them to `TinyGPSPlus`.

🌍 (d) Print GPS Coordinates

```
if (gps.location.isUpdated()) {
  Serial.print("Latitude: ");
  Serial.println(gps.location.lat(), 6);
  Serial.print("Longitude: ");
  Serial.println(gps.location.lng(), 6);
}
```

- If the GPS data was updated, prints **latitude and longitude** with 6 decimal places.

□ (e) Delay

```
delay(1000);
```

- Adds a 1-second delay before repeating the loop to avoid spamming the serial monitor.