TECNOSENSE

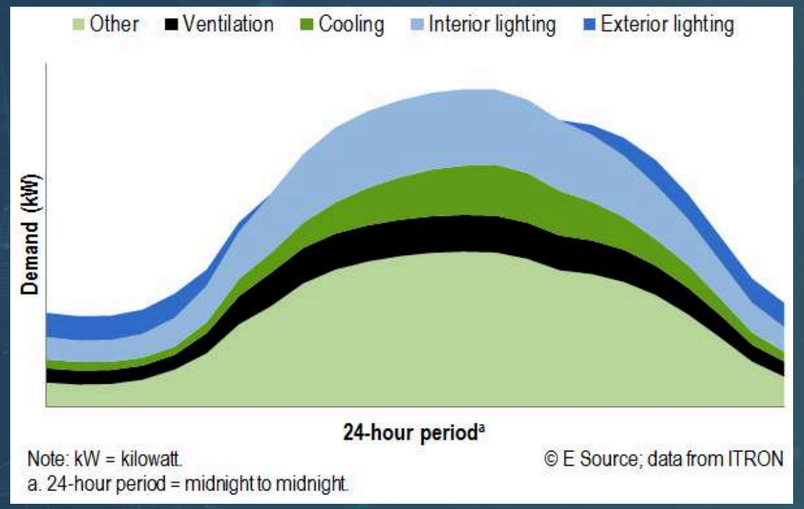
Intelligent Campus Automation and Space Optimization System

TEAM MEMBERS:
TEO SHENG JI
CHEANG TZE HAN
PEH KYUNG KHANG
SOH HUI YAO

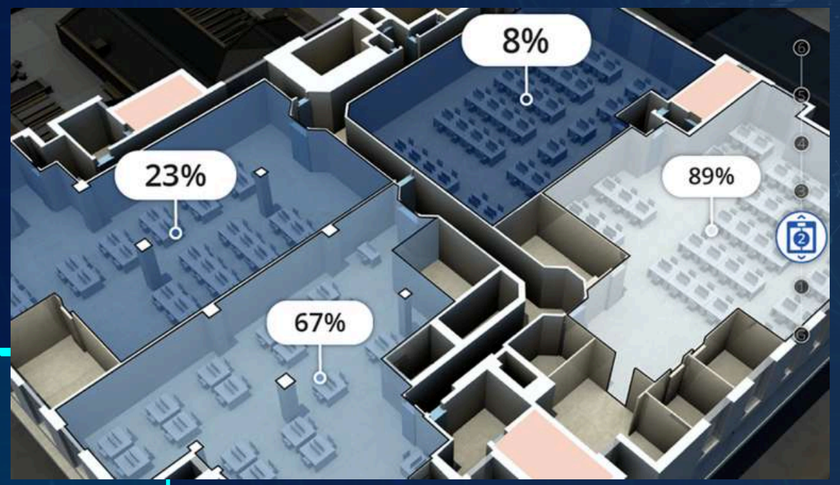
By team TECNOLOGIA

PROBLEM STATEMENT

- Inefficient energy usage in campus buildings



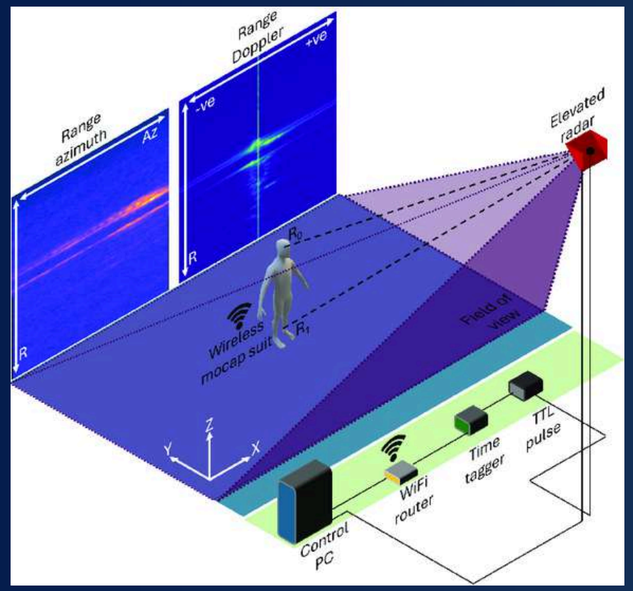
- Increased operational costs from unnecessary HVAC and lighting usage
- Limited analytics for facility management to make informed decisions

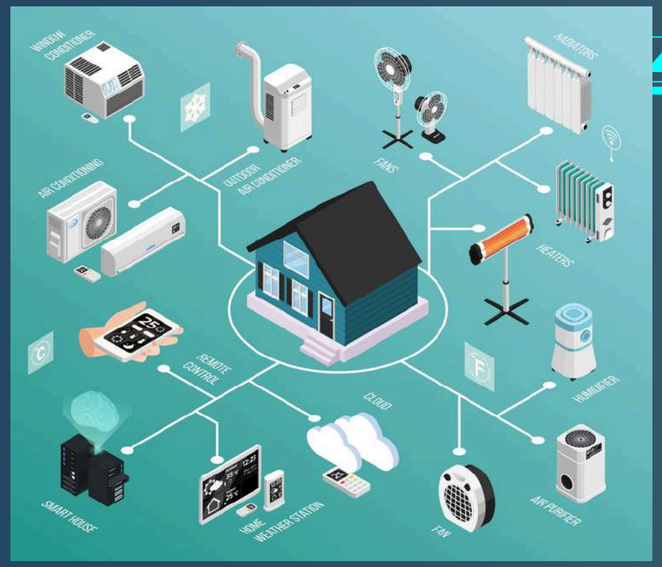


- Poor space utilization due to lack of real-time occupancy data

OBJECTIVE

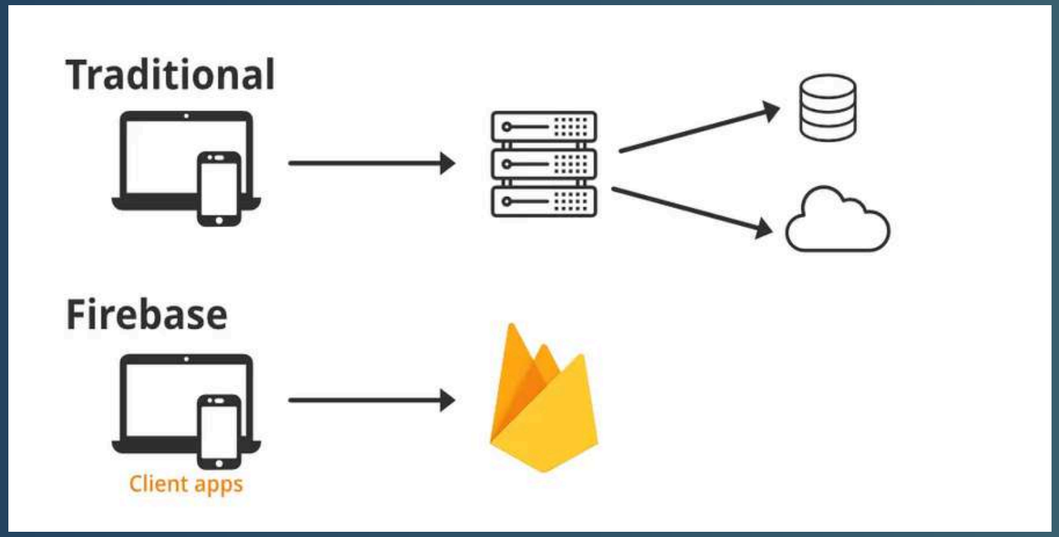
- Develop a light, reliable and cheap IoT-based automation system (NO CCTV/computer vision required)





- Utilize mmWave radar and environmental sensors for real-time monitoring

- Provide analytics for efficient space and operations management with real time controls



- Optimize lighting and HVAC systems to reduce energy waste

SYSTEM ARCHITECTURE OVERVIEW

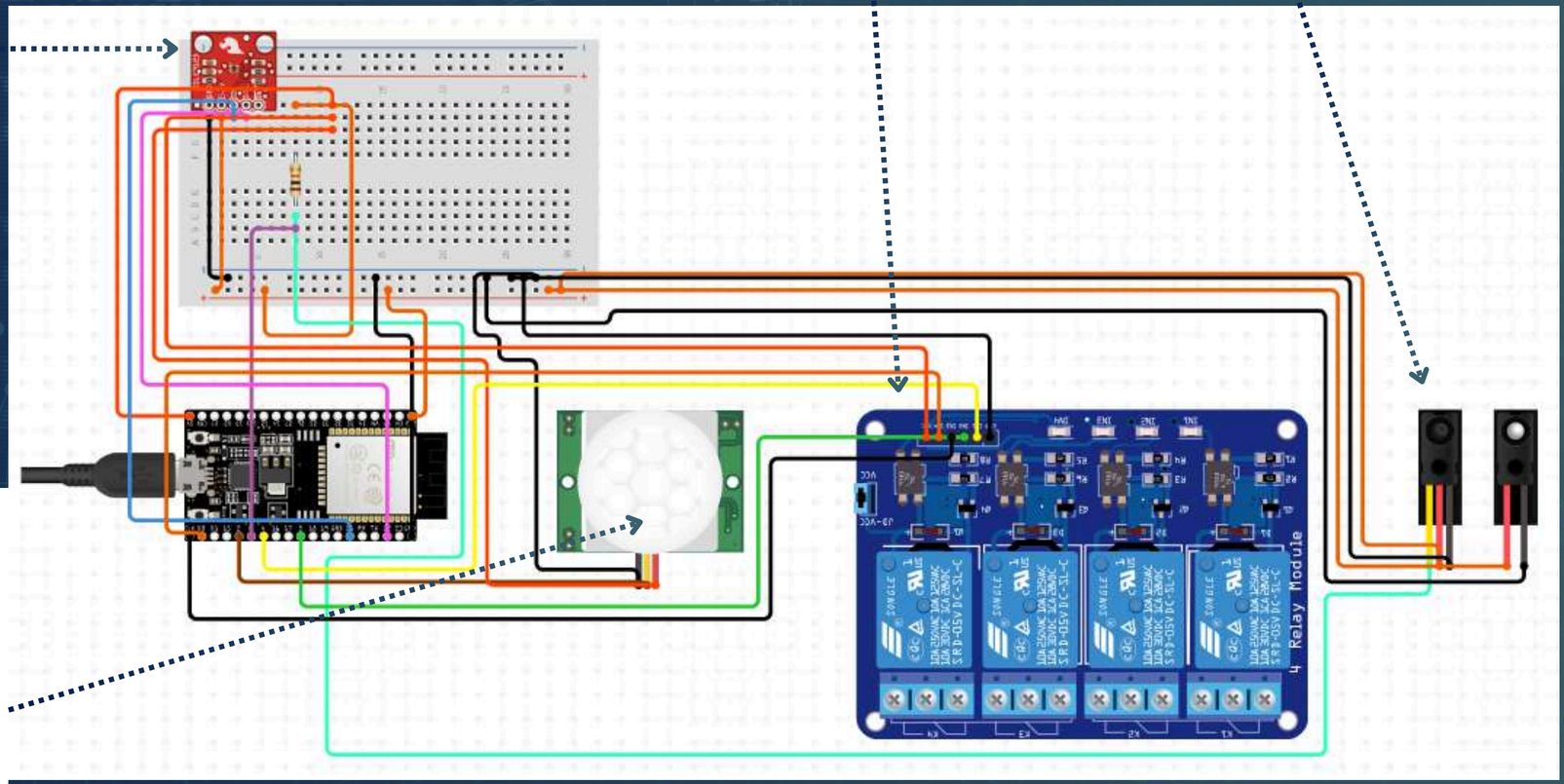
Hardware components

BH1750 ambient light sensor

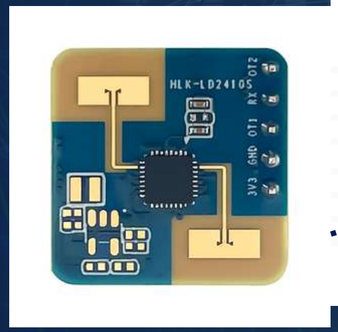
ESP32 with wifi/bluetooth

4-channel relay for output controls

Breakbeam IR (Doorway activation)



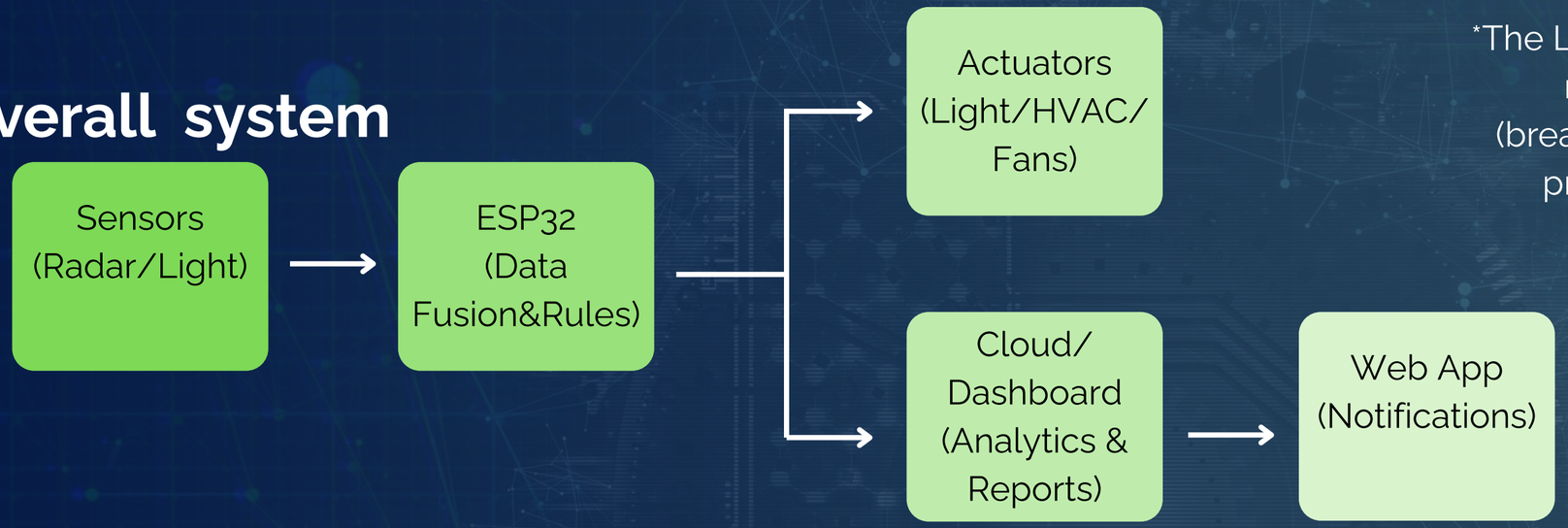
What is LD2410S?
*Measures Doppler shift (motion) and micro-Doppler patterns (heartbeat, breathing, slight movements).
Uses range-gated reflections to separate living stationary humans from background objects.



LD2410S mmWave radar in place of PIR sensor shown

HOW IT WORKS

Overall system



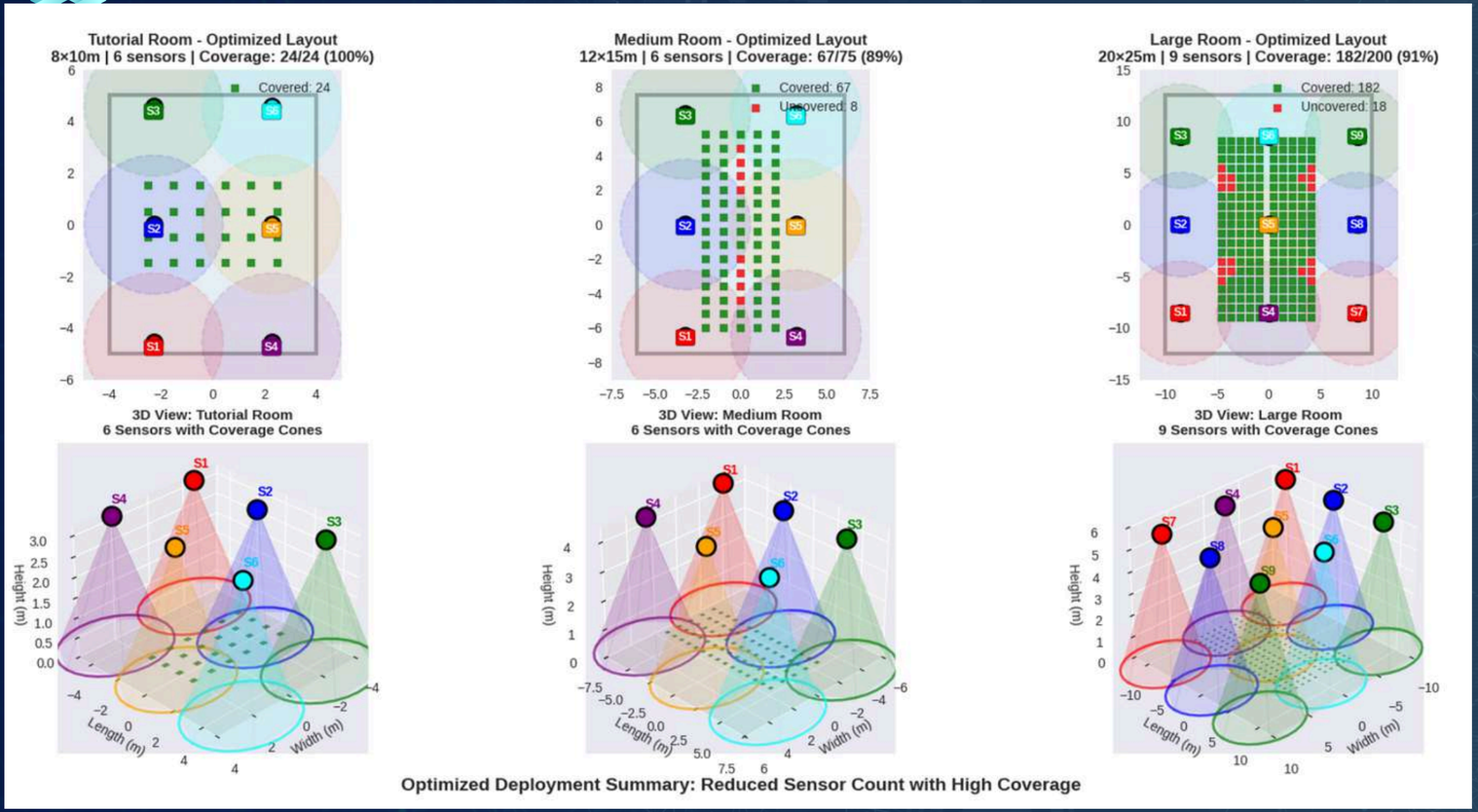
*The LD2410S mmWave radar detects humans by recognizing even tiny bio-movements (breathing/heartbeat), making it more reliable, private, and environment-proof than PIR, ultrasonic, or cameras.

1st layer: Breakbeam IR at doorway acts like activation/sleep layer for the system
It will only be switched on if theres any detection

2nd layer: The mmWave radar (LD2140S) starts detecting for any humans moving or static and power on those lights/fans/AC

*If IR detects something but no occupancy or other way around the detection is NOT VALID

SYSTEM ARCHITECTURE OVERVIEW



- Diagram above shows the number of sensors required to FULLY cover the area of rooms with NO CONSIDERATIONS for overlapping and cost cutting.

*Seating pattern matters more than sensor placement for maximum cost savings, notice those blindspots and overlapping waste.

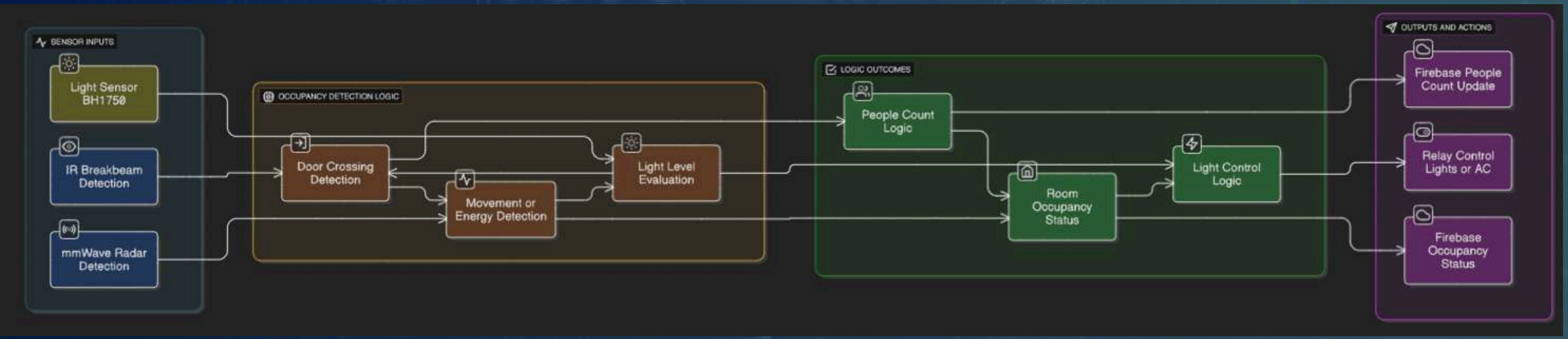
*The LD2410S mmWave radar detects humans by recognizing even tiny bio-movements (breathing/heartbeat), making it more reliable, private, and environment-proof than PIR, ultrasonic, or cameras.

- Number of sensors needed and their positions are mainly affected by seatings arrangement.
- Cost savings can be amended based on classroom seatings

WHY USE LD2410S?

Parameter	LD2410S mmWave	PIR Sensor
Detection Range	Up to **8 m**	**3–5 m** typical
Field of View	**~80° cone**	**~110° pyramid**
Detection Type	**Motion + Stationary**	**Motion only**
Accuracy	**>98%**	**70–80%**
Environmental	**All lighting conditions**	**Heat/light affected**
False Positives	**<2%**	**15–25%**
Power Consumption	**~150 mW**	**~50 mW**

SYSTEM ARCHITECTURE OVERVIEW



SYSTEM ARCHITECTURE OVERVIEW

```
#include <WiFi.h>
#include <FirebaseESP32.h>
#include <Wire.h>
#include <BH1750.h>
#include <ArduinoJson.h>
#include <NTPClient.h>
#include <WiFiUdp.h>

// ================== CONFIGURATION ==================
const char* WIFI_SSID = "CAMPUS_WIFI";
const char* WIFI_PASSWORD = "WIFI_PASSWORD";

// Firebase Configuration
#define FIREBASE_HOST "your-project.firebaseio.com"
#define FIREBASE_AUTH "your-database-secret"

// Room Identification
const char* BUILDING = "building_1";
const char* ROOM = "room_203";
const char* DEVICE_ID = "esp32_smart_node_1";

// Hardware Pins
#define RELAY_LIGHT 2
#define RELAY_AC 3
#define RELAY_FAN 4
#define IR_BREAKBEAM_PIN 1
#define RADAR_RX_PIN 43
#define RADAR_TX_PIN 44

// Sensor Thresholds
#define LUX_THRESHOLD 200
#define RADAR_ENERGY_THRESHOLD 50
#define MOVEMENT_TIMEOUT 15000        // 15 seconds
#define DOOR_DEBOUNCE_TIME 300        // 300ms
#define OCCUPANCY_TIMEOUT 60000       // 1 minute
#define MAX_PEOPLE 20

// ================== GLOBAL OBJECTS ==================
FirebaseData firebaseData;
FirebaseConfig config;
FirebaseAuth auth;
BH1750 lightSensor;
HardwareSerial radarSerial(1);
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org");

// ================== DATA STRUCTURES ==================
struct RoomStatus {
  // Occupancy
  int peopleCount = 0;
  bool isOccupied = false;

  // Devices
  bool lightOn = false;
  bool acOn = false;
  bool fanOn = false;
  bool autoMode = true;

  // Environment
  float lightLevel = 0;
  unsigned long lastActivity = 0;
};

struct SensorData {
  float lux = 0;
  bool radarPresence = false;
  bool doorBeamBroken = false;
  int radarEnergyLevel = 0;
  unsigned long timestamp = 0;
};

struct OccupancyLogic {
  // State tracking
  bool pendingEntry = false;
  bool pendingExit = false;
  unsigned long entryStartTime = 0;
  unsigned long exitStartTime = 0;

  // Timing
  unsigned long lastMovementTime = 0;
  unsigned long lastDoorEventTime = 0;
  bool lastBeamState = false;
};

// ================== GLOBAL VARIABLES ==================
RoomStatus room;
SensorData sensors;
OccupancyLogic occupancy;

// Firebase paths
String basePath;
String controlPath;
String statusPath;
String sensorsPath;

// Timing
unsigned long lastFirebaseUpdate = 0;
unsigned long lastControlCheck = 0;
volatile bool doorEventTriggered = false;
volatile unsigned long lastIrTime = 0;

// System status
bool radarConnected = false;
bool firebaseConnected = false;

// ================== INTERRUPT HANDLER ==================
void IRAM_ATTR doorSensorISR() {
  unsigned long currentTime = millis();
  if (currentTime - lastIrTime > DOOR_DEBOUNCE_TIME) {
    doorEventTriggered = true;
    lastIrTime = currentTime;
  }
}

// ================== INITIALIZATION ==================
void initializeSystem() {
  Serial.begin(115200);
  delay(1000);

  Serial.println("\n=== Smart Room Controller v3.0 ===");
  Serial.printf("Room: %s/%s | Device: %s\n", BUILDING, ROOM, DEVICE_ID);

  // Initialize Firebase paths
  basePath = String("/rooms/") + BUILDING + "/" + ROOM;
  controlPath = basePath + "/controls";
  statusPath = basePath + "/status";
  sensorsPath = basePath + "/sensors";

  initializeHardware();
  initializeWiFi();
  initializeFirebase();
  initializeTime();

  Serial.println("=== System Ready ===");
}

void initializeHardware() {
  Serial.println("Initializing hardware...");

  // Relays
  pinMode(RELAY_LIGHT, OUTPUT);
  pinMode(RELAY_AC, OUTPUT);
  pinMode(RELAY_FAN, OUTPUT);
  setAllRelays(false);

  // Light sensor
  Wire.begin(8, 9);
  if (lightSensor.begin(BH1750::CONTINUOUS_HIGH_RES_MODE)) {
    Serial.println("✓ Light sensor ready");
  } else {
    Serial.println("✗ Light sensor failed");
  }

  // Radar sensor
  initializeRadar();
```

SYSTEM ARCHITECTURE OVERVIEW

```
  // IR sensor
  pinMode(IR_BREAKBEAM_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(IR_BREAKBEAM_PIN), doorSensorISR, CHANGE);

  Serial.println("✓ Hardware initialized");
}

void initializeRadar() {
  radarSerial.begin(256000, SERIAL_8N1, RADAR_RX_PIN, RADAR_TX_PIN);
  delay(200);

  // Enable radar
  uint8_t enableCmd[] = {0xFD, 0xFC, 0xFB, 0xFA, 0x04, 0x00, 0xFF, 0x00, 0x01, 0x00, 0x04, 0x03, 0x02, 0x01};
  radarSerial.write(enableCmd, sizeof(enableCmd));
  delay(100);

  radarConnected = true;
  Serial.println("✓ Radar initialized");
}

void initializeWiFi() {
  Serial.print("Connecting to WiFi");
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 30) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.printf("\n✓ WiFi connected: %s\n", WiFi.localIP().toString().c_str());
  } else {
    Serial.println("\nX WiFi connection failed");
  }
}

void initializeFirebase() {
  if (WiFi.status() != WL_CONNECTED) return;

  config.host = FIREBASE_HOST;
  config.signer.tokens.legacy_token = FIREBASE_AUTH;

  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);

  // Test connection
  if (Firebase.ready()) {
    firebaseConnected = true;
    Serial.println("✓ Firebase connected");

    // Initialize device status in Firebase
    updateFirebaseStatus();
  } else {
    Serial.println("X Firebase connection failed");
  }
}

void initializeTime() {
  timeClient.begin();
  timeClient.setTimeOffset(28800); // GMT+8 for Malaysia
  Serial.println("✓ Time client initialized");
}

// ================== SENSOR READING ==================
void readAllSensors() {
  sensors.timestamp = millis();

  // Light sensor
  sensors.lux = lightSensor.readLightLevel();
  if (sensors.lux < 0) sensors.lux = 0;
  room.lightLevel = sensors.lux;

  // Radar sensor
  readRadarSensor();

  // Door beam sensor
  sensors.doorBeamBroken = (digitalRead(IR_BREAKBEAM_PIN) == LOW);
}

void readRadarSensor() {
  static unsigned long lastRadarRead = 0;

  if (!radarConnected || (millis() - lastRadarRead < 100)) return;
  lastRadarRead = millis();

  sensors.radarEnergyLevel = 0;
  sensors.radarPresence = false;

  int bytesAvailable = radarSerial.available();
  if (bytesAvailable > 10) {
    uint8_t buffer[64];
    int bytesRead = min(bytesAvailable, 64);

    // Read data
    for (int i = 0; i < bytesRead && radarSerial.available(); i++) {
      buffer[i] = radarSerial.read();
    }

    // Parse radar data
    parseRadarData(buffer, bytesRead);
  }
}
```

SYSTEM ARCHITECTURE OVERVIEW

```
263  void parseRadarData(uint8_t* buffer, int length) {
264    for (int i = 0; i < length - 8; i++) {
265      // Look for data frame header
266      if (buffer[i] == 0xF4 && buffer[i+1] == 0xF3 &&
267          buffer[i+2] == 0xF2 && buffer[i+3] == 0xF1) {
268
269        int energyValue = (buffer[i+6] << 8) | buffer[i+5];
270        sensors.radarEnergyLevel = max(sensors.radarEnergyLevel, energyValue);
271
272        if (energyValue > RADAR_ENERGY_THRESHOLD) {
273          sensors.radarPresence = true;
274        }
275        break;
276      }
277    }
278
279    // Fallback: detect presence by serial activity
280    if (!sensors.radarPresence && radarSerial.available() > 20) {
281      sensors.radarPresence = true;
282      sensors.radarEnergyLevel = 100;
283    }
284  }
285
286  // ================== OCCUPANCY DETECTION ==================
287  void processOccupancyDetection() {
288    unsigned long currentTime = millis();
289
290    // Handle door events
291    if (doorEventTriggered) {
292      handleDoorEvent(currentTime);
293      doorEventTriggered = false;
294    }
295
296    // Update movement tracking
297    if (sensors.radarPresence) {
298      occupancy.lastMovementTime = currentTime;
299      room.lastActivity = currentTime;
300    }
301
302    // Process door crossing logic
303    updateDoorCrossingLogic(currentTime);
304
305    // Handle occupancy timeout
306    handleOccupancyTimeout(currentTime);
307
308    // Update room status
309    room.isOccupied = (room.peopleCount > 0) ||
310                      (currentTime - occupancy.lastMovementTime < MOVEMENT_TIMEOUT);
311  }
312
```

```
313  void handleDoorEvent(unsigned long currentTime) {
314    occupancy.lastDoorEventTime = currentTime;
315    occupancy.lastBeamState = sensors.doorBeamBroken;
316
317    Serial.printf("Door: %s | Radar: %s (Energy: %d)\n",
318                  sensors.doorBeamBroken ? "BLOCKED" : "CLEAR",
319                  sensors.radarPresence ? "DETECTED" : "NONE",
320                  sensors.radarEnergyLevel);
321  }
322
323  void updateDoorCrossingLogic(unsigned long currentTime) {
324    // Start entry/exit detection
325    if (sensors.doorBeamBroken && sensors.radarPresence &&
326        !occupancy.pendingEntry && !occupancy.pendingExit) {
327
328      if (shouldTriggerEntry()) {
329        startEntrySequence(currentTime);
330      } else {
331        startExitSequence(currentTime);
332      }
333    }
334
335    // Complete entry
336    if (occupancy.pendingEntry && !sensors.doorBeamBroken) {
337      if (isValidCrossingDuration(currentTime - occupancy.entryStartTime)) {
338        completeEntry();
339      }
340      occupancy.pendingEntry = false;
341    }
342
343    // Complete exit
344    if (occupancy.pendingExit && !sensors.doorBeamBroken) {
345      if (isValidCrossingDuration(currentTime - occupancy.exitStartTime)) {
346        completeExit();
347      }
348      occupancy.pendingExit = false;
349    }
350
351    // Clear stuck states
352    clearStuckStates(currentTime);
353  }
354
355  bool shouldTriggerEntry() {
356    // Entry more likely if room is empty or strong radar signal
357    return (room.peopleCount == 0) ||
358           (sensors.radarEnergyLevel > RADAR_ENERGY_THRESHOLD * 1.5);
359  }
360
361  void startEntrySequence(unsigned long currentTime) {
362    occupancy.pendingEntry = true;
363    occupancy.entryStartTime = currentTime;
364    Serial.println("→ Entry sequence started");
365  }
366
```

```
367  void startExitSequence(unsigned long currentTime) {
368    occupancy.pendingExit = true;
369    occupancy.exitStartTime = currentTime;
370    Serial.println("← Exit sequence started");
371  }
372
373  bool isValidCrossingDuration(unsigned long duration) {
374    return (duration > DOOR_DEBOUNCE_TIME && duration < 3000);
375  }
376
377  void completeEntry() {
378    room.peopleCount = min(room.peopleCount + 1, MAX_PEOPLE);
379    occupancy.lastMovementTime = millis();
380    Serial.printf("✓ Entry confirmed - Count: %d\n", room.peopleCount);
381  }
382
383  void completeExit() {
384    room.peopleCount = max(room.peopleCount - 1, 0);
385    Serial.printf("✓ Exit confirmed - Count: %d\n", room.peopleCount);
386  }
387
388  void clearStuckStates(unsigned long currentTime) {
389    const unsigned long STUCK_TIMEOUT = 5000;
390
391    if (occupancy.pendingEntry &&
392        (currentTime - occupancy.entryStartTime > STUCK_TIMEOUT)) {
393      occupancy.pendingEntry = false;
394      Serial.println("Entry timeout cleared");
395    }
396
397    if (occupancy.pendingExit &&
398        (currentTime - occupancy.exitStartTime > STUCK_TIMEOUT)) {
399      occupancy.pendingExit = false;
400      Serial.println("Exit timeout cleared");
401    }
402  }
403
404  void handleOccupancyTimeout(unsigned long currentTime) {
405    if (room.peopleCount > 0 &&
406        (currentTime - occupancy.lastMovementTime > OCCUPANCY_TIMEOUT)) {
407      Serial.println("⚠ Occupancy timeout - clearing room");
408      room.peopleCount = 0;
409      room.isOccupied = false;
410    }
411  }
412
413  // ================== AUTOMATION CONTROL ==================
414  void applyAutomationLogic() {
415    if (!room.autoMode) return;
416
417    controlLighting();
418    controlFan();
419    controlAirConditioning();
420  }
421
```

SYSTEM ARCHITECTURE OVERVIEW

```
void controlLighting() {
  static unsigned long lightOffDelay = 0;
  bool shouldBeOn = room.isOccupied && (sensors.lux < LUX_THRESHOLD);

  if (shouldBeOn && !room.lightOn) {
    setLight(true);
    lightOffDelay = 0;
    Serial.println("Auto: Light ON");
  }
  else if (!shouldBeOn && room.lightOn) {
    // Delayed turn off to prevent flickering
    if (lightOffDelay == 0) {
      lightOffDelay = millis();
    } else if (millis() - lightOffDelay > 10000) {
      setLight(false);
      lightOffDelay = 0;
      Serial.println("Auto: Light OFF");
    }
  }
  else if (shouldBeOn) {
    lightOffDelay = 0; // Cancel delayed off
  }
}

void controlFan() {
  bool shouldBeOn = room.isOccupied;

  if (shouldBeOn != room.fanOn) {
    setFan(shouldBeOn);
    Serial.printf("Auto: Fan %s\n", shouldBeOn ? "ON" : "OFF");
  }
}

void controlAirConditioning() {
  bool shouldBeOn = (room.peopleCount >= 3); // AC for 3+ people

  if (shouldBeOn != room.acOn) {
    setAC(shouldBeOn);
    Serial.printf("Auto: AC %s (Count: %d)\n",
                  shouldBeOn ? "ON" : "OFF", room.peopleCount);
  }
}

// ================== DEVICE CONTROL ==================
void setLight(bool state) {
  room.lightOn = state;
  digitalWrite(RELAY_LIGHT, state ? HIGH : LOW);
}

void setFan(bool state) {
  room.fanOn = state;
  digitalWrite(RELAY_FAN, state ? HIGH : LOW);
}

void setAC(bool state) {
  room.acOn = state;
  digitalWrite(RELAY_AC, state ? HIGH : LOW);
```

```
void setAllRelays(bool state) {
  setLight(state);
  setFan(state);
  setAC(state);
}

// ================== FIREBASE COMMUNICATION ==================
void updateFirebase() {
  if (!firebaseConnected || !Firebase.ready()) return;

  unsigned long currentTime = millis();
  if (currentTime - lastFirebaseUpdate < 5000) return; // Limit updates

  updateFirebaseSensors();
  updateFirebaseStatus();

  lastFirebaseUpdate = currentTime;
}

void updateFirebaseSensors() {
  FirebaseJson sensorJson;
  timeClient.update();

  sensorJson.set("timestamp", timeClient.getEpochTime());
  sensorJson.set("lux", sensors.lux);
  sensorJson.set("peopleCount", room.peopleCount);
  sensorJson.set("isOccupied", room.isOccupied);
  sensorJson.set("radarPresence", sensors.radarPresence);
  sensorJson.set("radarEnergy", sensors.radarEnergyLevel);
  sensorJson.set("doorBlocked", sensors.doorBeamBroken);

  Firebase.pushJSON(firebaseData, sensorsPath.c_str(), sensorJson);
}

void updateFirebaseStatus() {
  FirebaseJson statusJson;

  statusJson.set("deviceId", DEVICE_ID);
  statusJson.set("autoMode", room.autoMode);
  statusJson.set("lightOn", room.lightOn);
  statusJson.set("acOn", room.acOn);
  statusJson.set("fanOn", room.fanOn);
  statusJson.set("wifiConnected", WiFi.status() == WL_CONNECTED);
  statusJson.set("wifiRSSI", WiFi.RSSI());
  statusJson.set("freeHeap", ESP.getFreeHeap());
  statusJson.set("uptime", millis() / 1000);
  statusJson.set("lastUpdate", timeClient.getEpochTime());

  Firebase.setJSON(firebaseData, statusPath.c_str(), statusJson);
}
```

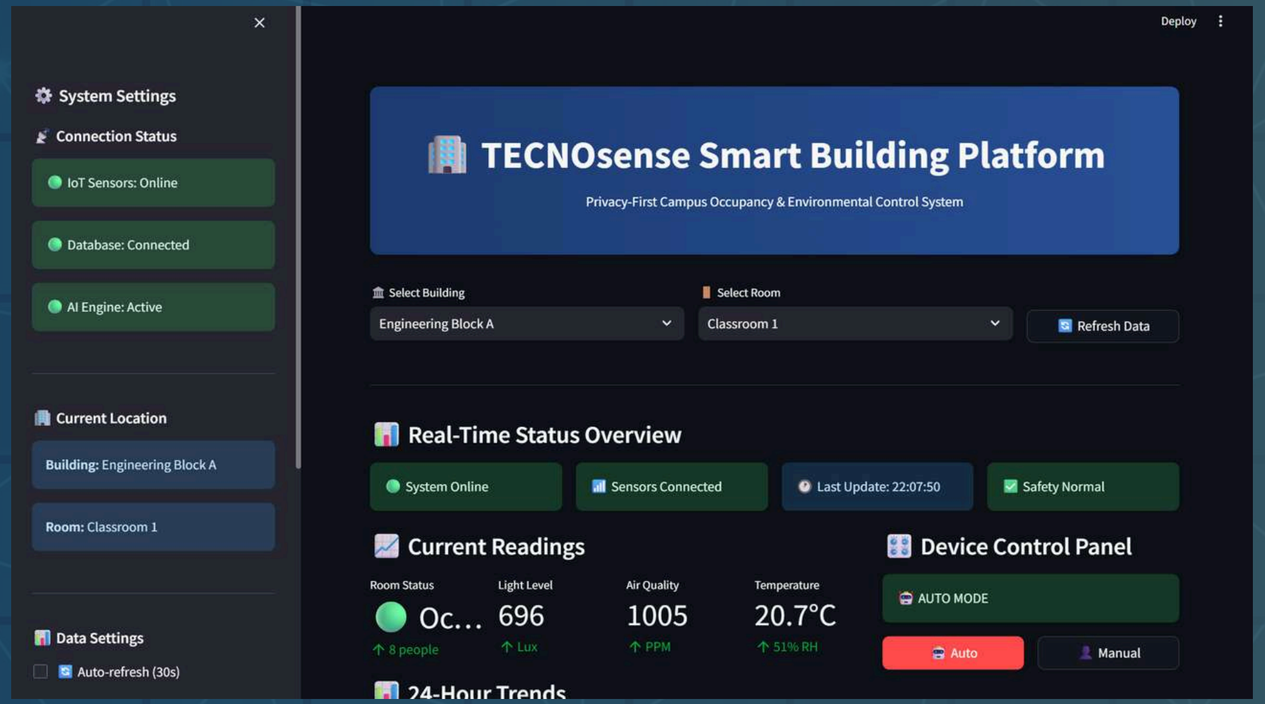
```
void checkFirebaseControls() {
  if (!firebaseConnected || !Firebase.ready()) return;

  unsigned long currentTime = millis();
  if (currentTime - lastControlCheck < 2000) return; // Check every 2 seco

  if (Firebase.getJSON(firebaseData, controlPath.c_str())) {
    FirebaseJson json = firebaseData.jsonObject();
    FirebaseJsonData jsonData;

    // Check light control
    if (json.get(jsonData, "light") && jsonData.success) {
      bool lightState = jsonData.boolValue;
      if (lightState != room.lightOn) {
        setLight(lightState);
        room.autoMode = false; // Disable auto mode on manual control
        Serial.printf("Remote: Light %s\n", lightState ? "ON" : "OFF");
      }
    }

    // Check fan control
    if (json.get(jsonData, "fan") && jsonData.success) {
      bool fanState = jsonData.boolValue;
      if (fanState != room.fanOn) {
        setFan(fanState);
        room.autoMode = false;
        Serial.printf("Remote: Fan %s\n", fanState ? "ON" : "OFF");
      }
    }

    // Check AC control
    if (json.get(jsonData, "ac") && jsonData.success) {
      bool acState = jsonData.boolValue;
      if (acState != room.acOn) {
        setAC(acState);
        room.autoMode = false;
        Serial.printf("Remote: AC %s\n", acState ? "ON" : "OFF");
      }
    }

    // Check auto mode
    if (json.get(jsonData, "autoMode") && jsonData.success) {
      bool autoState = jsonData.boolValue;
      if (autoState != room.autoMode) {
        room.autoMode = autoState;
        Serial.printf("Remote: Auto mode %s\n", autoState ? "ON" : "OFF");
      }
    }
  }

  lastControlCheck = currentTime;
}
```

PROTOTYPE COST

Component	Price
ESP32	RM17
LD2410S (quantity depends on size of room)	RM18
Breakbeam IR	RM10
BH1750 (light sensor)	RM4.5
Relay module (4 channel)	RM6.6
Total prototype cost: RM56.1	

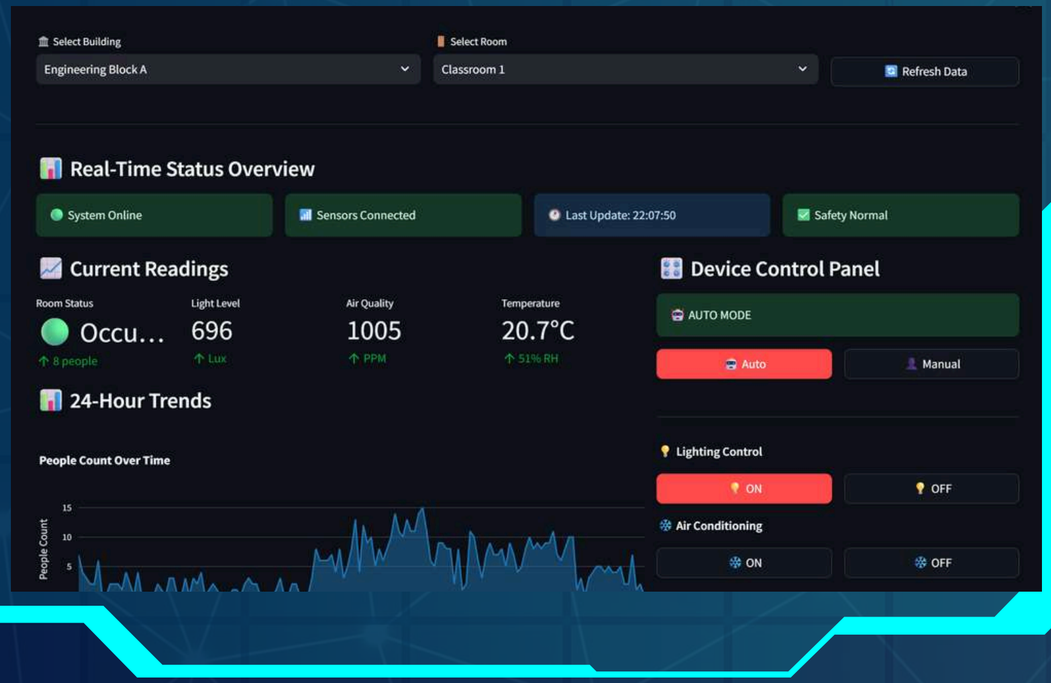
2. COMMUNICATION PROTOCOL

- The system leverages MQTT, a lightweight publish-subscribe protocol, for efficient, low-overhead communication.
- It connects via Wi-Fi 6, enabling fast, campus-wide coverage with seamless roaming.
- All transmissions are secured with TLS encryption to ensure data integrity and confidentiality.



SOFTWARE UI

*Program codes shown in github



SDG

7

Affordable and Clean Energy

- TECNOsense reduces energy waste by 30–50% in lighting and HVAC.
- Promotes efficient energy management through IoT and automation.

9

Industry, Innovation, and Infrastructure

- Modernizes campus infrastructure with IoT-based smart systems.
- Uses mmWave radar + sensor fusion for precision and innovation

12

Responsible Consumption and Production

- Avoids energy over consumption by linking usage to real occupancy.
- Data analytics help optimize schedules and reduce waste.

LIMITATIONS AND FUTURE IMPROVEMENTS

Limited controls and less dynamic.	Include more components for light dimming, AC temperature controls and power monitoring with AI
Lacks large scale integration between units	Develop a stronger backend and database for large scale space management in campus with efficient utilisation and booking features

CONCLUSIONS

TECNOSENSE CORE

- 2-layer verification occupancy detection. (IR +radar)
- Environment based lighting switch
- Real life monitoring and control with database

OUTCOME

- Simplification and cost efficiency
- Power and space optimisation
- Reliable and comfortable to use