

XBio Sentinel Patent Filing

===== XBio
Sentinel - Source Code Extract (SAIP) للتقديم لهيئة الملكية الفكرية السعودية
=====

تاريخ التقديم: فبراير 2026
رقم طلب البراءة: SA 1020258841
MR.F@MRF103.COM

===== وصف المنتج
=====

(Offline-AI) هو نظام استشعار بيئي ذكي يستخدم الذكاء الاصطناعي المحلي XBio Sentinel لتحليل جودة الهواء وتصنيف الروائح. يتميز بالعمل بدون اتصال بالإنترنت مع قدرات تعلم تكيفية.

المكونات: - Hardware: ESP32-S3 N16R8 + BME688 (Bosch) Sensor -
Firmware: C++ / Arduino Framework / PlatformIO - Mobile App:
React + Capacitor (Android APK) - Backend API: Node.js + Express +
PostgreSQL

===== 1.
ESP32 FIRMWARE (main.cpp)
=====

/** * XBio Sentinel - ESP32-S3 Main Firmware * Copyright (c) 2026
ARC Advanced Environmental Technologies * Patent Pending: SA
1020258841 */

```
#include <Arduino.h> #include <WiFi.h> #include <BLEDevice.h>  
#include <BLEServer.h> #include <BLE2902.h> #include <Wire.h>  
#include "bme688_driver.h" #include "smell_classifier.h" #include  
"config.h"
```

```
// BLE Configuration #define SERVICE_UUID "4fafc201-1fb5-459e-  
8fcc-c5c9c331914b" #define CHARACTERISTIC_UUID "beb5483e-  
36e1-4688-b7f5-ea07361b26a8" #define DEVICE_NAME "XBio-  
Sentinel"
```

```
// Global Objects BME688Driver bme688; SmellClassifier classifier;  
BLEServer* pServer = nullptr; BLECharacteristic* pCharacteristic =  
nullptr; bool deviceConnected = false;
```

```
// Sensor Data Structure struct SensorData { float temperature; float  
humidity; float pressure; float gasResistance; float iaq; // Indoor Air  
Quality Index float voc; // Volatile Organic Compounds float eCO2; //  
Estimated CO2 char smellClass[32]; // AI Classification Result float  
confidence; // Classification Confidence uint32_t timestamp; };
```

```
SensorData currentReading;
```

```

// BLE Callbacks class ServerCallbacks: public BLEServerCallbacks {
void onConnect(BLEServer* pServer) { deviceConnected = true;
Serial.println("BLE Client Connected"); }

void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
    Serial.println("BLE Client Disconnected");
    BLEDevice::startAdvertising();
}

};

void initBLE() { BLEDevice::init(DEVICE_NAME); pServer =
BLEDevice::createServer(); pServer->setCallbacks(new
ServerCallbacks());

BLEService* pService = pServer->createService(SERVICE_UUID);
pCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_READ |
    BLECharacteristic::PROPERTY_NOTIFY
);
pCharacteristic->addDescriptor(new BLE2902());
pService->start();

BLEAdvertising* pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pAdvertising->setScanResponse(true);
pAdvertising->setMinPreferred(0x06);
BLEDevice::startAdvertising();

Serial.println("BLE Advertising Started");
}

void initSensors() { Wire.begin(SDA_PIN, SCL_PIN);

if (!bme688.begin()) {
    Serial.println("BME688 initialization failed!");
    while (1) delay(1000);
}

// Configure BME688 for gas sensing
bme688.setTemperatureOversampling(BME688_OS_8X);
bme688.setHumidityOversampling(BME688_OS_2X);
bme688.setPressureOversampling(BME688_OS_4X);
bme688.setIIRFilterSize(BME688_FILTER_SIZE_3);
bme688.setGasHeater(320, 150); // 320°C for 150ms

Serial.println("BME688 Initialized Successfully");
}

void readSensors() { if (bme688.performReading()) {
currentReading.temperature = bme688.temperature;
currentReading.humidity = bme688.humidity;
currentReading.pressure = bme688.pressure / 100.0; // Convert to
hPa currentReading.gasResistance = bme688.gas_resistance / 1000.0;
// kOhms

    // Calculate derived values
    currentReading.iaq = bme688.calculateIAQ();

```

```

    currentReading.voc = bme688.calculateVOC();
    currentReading.eCO2 = bme688.calculateECO2();

    // AI Smell Classification (Offline)
    float features[4] = {
        currentReading.temperature,
        currentReading.humidity,
        currentReading.gasResistance,
        currentReading.iaq
    };

    classifier.classify(features,
                        currentReading.smellClass,
                        &currentReading.confidence);

    currentReading.timestamp = millis();
}

}

String formatJSON() { String json = "{"; json += "\"temp\": " +
String(currentReading.temperature, 2) + ","; json += "\"hum\": " +
String(currentReading.humidity, 2) + ","; json += "\"pres\": " +
String(currentReading.pressure, 2) + ","; json += "\"gas\": " +
String(currentReading.gasResistance, 2) + ","; json += "\"iaq\": " +
String(currentReading.iaq, 1) + ","; json += "\"voc\": " +
String(currentReading.voc, 2) + ","; json += "\"eco2\": " +
String(currentReading.eCO2, 0) + ","; json += "\"smell\": " +
String(currentReading.smellClass) + ","; json += "\"conf\": " +
String(currentReading.confidence, 3) + ","; json += "\"ts\": " +
String(currentReading.timestamp); json += "}"; return json; }

void sendBLEData() { if (deviceConnected) { String jsonData =
formatJSON(); pCharacteristic->setValue(jsonData.c_str());
pCharacteristic->notify(); } }

void setup() { Serial.begin(115200); Serial.println("=== XBio
Sentinel Starting ===");

initSensors();
classifier.loadModel(); // Load offline AI model
initBLE();

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

}

void loop() { readSensors(); sendBLEData();

// Debug output
Serial.printf("T: %.1f°C H: %.1f%% P: %.1f hPa Gas: %.1f kΩ IAQ: %.0f\n",
    currentReading.temperature,
    currentReading.humidity,
    currentReading.pressure,
    currentReading.gasResistance,
    currentReading.iaq
);
Serial.printf("Smell: %s (%.1f%% confidence)\n",
    currentReading.smellClass,
    currentReading.confidence * 100
);
}

```

```

delay(READING_INTERVAL_MS);

}

=====
===== 2.
BME688 DRIVER (bme688_driver.h)
=====
=====

/** * BME688 Gas Sensor Driver for XBio Sentinel * Provides
environmental sensing and gas resistance measurement */

#ifndef BME688_DRIVER_H #define BME688_DRIVER_H

#include <Wire.h>

// BME688 I2C Address #define BME688_I2C_ADDR 0x76

// Oversampling settings #define BME688_OS_NONE 0 #define
BME688_OS_1X 1 #define BME688_OS_2X 2 #define BME688_OS_4X
3 #define BME688_OS_8X 4 #define BME688_OS_16X 5

// Filter sizes #define BME688_FILTER_SIZE_0 0 #define
BME688_FILTER_SIZE_1 1 #define BME688_FILTER_SIZE_3 2
#define BME688_FILTER_SIZE_7 3 #define
BME688_FILTER_SIZE_15 4 #define BME688_FILTER_SIZE_31 5
#define BME688_FILTER_SIZE_63 6 #define
BME688_FILTER_SIZE_127 7

class BME688Driver { public: float temperature; float humidity; float
pressure; float gas_resistance;

bool begin(TwoWire *wire = &Wire) {
    _wire = wire;

    // Check chip ID
    uint8_t chipId = readRegister(0xD0);
    if (chipId != 0x61) {
        return false;
    }

    // Soft reset
    writeRegister(0xE0, 0xB6);
    delay(10);

    // Read calibration data
    readCalibrationData();

    return true;
}

void setTemperatureOversampling(uint8_t os) {
    uint8_t ctrl = readRegister(0x74);
    ctrl = (ctrl & 0x1F) | (os << 5);
    writeRegister(0x74, ctrl);
}

void setHumidityOversampling(uint8_t os) {
    writeRegister(0x72, os);
}

```

```

void setPressureOversampling(uint8_t os) {
    uint8_t ctrl = readRegister(0x74);
    ctrl = (ctrl & 0xE3) | (os << 2);
    writeRegister(0x74, ctrl);
}

void setIIRFilterSize(uint8_t size) {
    uint8_t config = readRegister(0x75);
    config = (config & 0xE3) | (size << 2);
    writeRegister(0x75, config);
}

void setGasHeater(uint16_t temp, uint16_t duration) {
    _heaterTemp = temp;
    _heaterDuration = duration;

    // Calculate heater resistance
    uint8_t res = calculateHeaterResistance(temp);
    writeRegister(0x5A, res);

    // Set heater duration
    uint8_t dur = calculateHeaterDuration(duration);
    writeRegister(0x64, dur);

    // Enable gas measurement
    uint8_t ctrl = readRegister(0x71);
    ctrl |= 0x10; // run_gas = 1
    writeRegister(0x71, ctrl);
}

bool performReading() {
    // Trigger forced mode measurement
    uint8_t ctrl = readRegister(0x74);
    ctrl = (ctrl & 0xFC) | 0x01; // forced mode
    writeRegister(0x74, ctrl);

    // Wait for measurement
    delay(_heaterDuration + 50);

    // Check if new data available
    uint8_t status = readRegister(0x1D);
    if (!(status & 0x80)) {
        return false; // No new data
    }

    // Read raw data
    uint8_t data[15];
    readRegisters(0x1F, data, 15);

    // Parse temperature
    uint32_t adc_temp = ((uint32_t)data[5] << 12) |
                        ((uint32_t)data[6] << 4) |
                        (data[7] >> 4);
    temperature = compensateTemperature(adc_temp);

    // Parse pressure
    uint32_t adc_pres = ((uint32_t)data[2] << 12) |
                        ((uint32_t)data[3] << 4) |
                        (data[4] >> 4);
    pressure = compensatePressure(adc_pres);
}

```

```

    // Parse humidity
    uint16_t adc_hum = ((uint16_t)data[8] << 8) | data[9];
    humidity = compensateHumidity(adc_hum);

    // Parse gas resistance
    uint16_t adc_gas = ((uint16_t)data[13] << 2) | (data[14] >> 6);
    uint8_t gas_range = data[14] & 0x0F;
    gas_resistance = calculateGasResistance(adc_gas, gas_range);

    return true;
}

float calculateIAQ() {
    // Indoor Air Quality Index (0-500)
    // Based on gas resistance and humidity compensation
    float hum_score = 0.25 * (humidity - 40.0) * (humidity - 40.0);
    float gas_score = log10(gas_resistance) * 10.0;
    float iaq = 500.0 - (hum_score + (100.0 - gas_score));
    return constrain(iaq, 0, 500);
}

float calculateVOC() {
    // VOC in ppm (parts per million)
    return 1000.0 / gas_resistance;
}

float calculateECO2() {
    // Estimated CO2 in ppm
    float voc = calculateVOC();
    return 400.0 + (voc * 100.0);
}

private: TwoWire* _wire; uint16_t _heaterTemp; uint16_t
_heaterDuration;

// Calibration coefficients
uint16_t par_t1;
int16_t par_t2;
int8_t par_t3;
// ... additional calibration parameters

void readCalibrationData() {
    // Read temperature calibration
    par_t1 = readRegister16(0xE9);
    par_t2 = (int16_t)readRegister16(0x8A);
    par_t3 = (int8_t)readRegister(0x8C);
    // ... read other calibration values
}

uint8_t readRegister(uint8_t reg) {
    _wire->beginTransmission(BME688_I2C_ADDR);
    _wire->write(reg);
    _wire->endTransmission();
    _wire->requestFrom(BME688_I2C_ADDR, (uint8_t)1);
    return _wire->read();
}

uint16_t readRegister16(uint8_t reg) {
    uint8_t lsb = readRegister(reg);
    uint8_t msb = readRegister(reg + 1);

```

```

        return ((uint16_t)msb << 8) | lsb;
    }

    void readRegisters(uint8_t reg, uint8_t* data, uint8_t len) {
        _wire->beginTransmission(BME688_I2C_ADDR);
        _wire->write(reg);
        _wire->endTransmission();
        _wire->requestFrom(BME688_I2C_ADDR, len);
        for (uint8_t i = 0; i < len; i++) {
            data[i] = _wire->read();
        }
    }

    void writeRegister(uint8_t reg, uint8_t value) {
        _wire->beginTransmission(BME688_I2C_ADDR);
        _wire->write(reg);
        _wire->write(value);
        _wire->endTransmission();
    }

    float compensateTemperature(uint32_t adc) {
        float var1 = ((float)adc / 16384.0 - (float)par_t1 / 1024.0) *
            (float)par_t2;
        float var2 = (((float)adc / 131072.0 - (float)par_t1 / 8192.0) *
            ((float)adc / 131072.0 - (float)par_t1 / 8192.0)) *
            ((float)par_t3 * 16.0);
        return (var1 + var2) / 5120.0;
    }

    float compensatePressure(uint32_t adc) {
        // Pressure compensation algorithm
        return (float)adc * 0.01; // Simplified
    }

    float compensateHumidity(uint16_t adc) {
        // Humidity compensation algorithm
        return (float)adc * 0.01; // Simplified
    }

    uint8_t calculateHeaterResistance(uint16_t temp) {
        return (uint8_t)((temp - 200) / 10);
    }

    uint8_t calculateHeaterDuration(uint16_t duration) {
        uint8_t factor = 0;
        uint8_t durval = 0;

        if (duration >= 0xFC0) {
            durval = 0xFF;
        } else {
            while (duration > 0x3F) {
                duration /= 4;
                factor++;
            }
            durval = (uint8_t)(duration + (factor * 64));
        }
        return durval;
    }

    float calculateGasResistance(uint16_t adc, uint8_t range) {

```

```

    const float lookup1[] = {1, 1, 1, 1, 1, 0.99, 1, 0.992, 1, 1,
0.998, 0.995, 1, 0.99, 1, 1};
    const float lookup2[] = {8000000, 4000000, 2000000, 1000000,
499500.4995, 248262.1648, 125000, 63004.03226, 31281.28128, 15625,
7812.5, 3906.25, 1953.125, 976.5625, 488.28125, 244.140625};

    float var1 = (1340.0 + (5.0 * 0)) * lookup1[range];
    float gas_res = var1 * lookup2[range] / (adc - 512.0 + var1);

    return gas_res;
}

};

#endif // BME688_DRIVER_H

=====
===== 3.
SMELL CLASSIFIER (smell_classifier.h)
=====
=====

/** * Offline AI Smell Classifier for XBio Sentinel * Uses embedded
neural network for smell classification */

#ifndef SMELL_CLASSIFIER_H #define SMELL_CLASSIFIER_H

#include <Arduino.h>

// Smell Categories const char* SMELL_CATEGORIES[] = { "neutral",
"fresh_air", "cooking", "smoke", "chemical", "organic_decay",
"perfume", "fuel", "moisture", "dust" }; const int NUM_CATEGORIES
= 10;

// Neural Network Weights (Embedded Model) // Input: [temperature,
humidity, gas_resistance, iaq] // Hidden Layer: 8 neurons // Output: 10
categories

class SmellClassifier { public: bool loadModel() { // Initialize weights
(pre-trained model embedded) initializeWeights(); _modelLoaded =
true; Serial.println("AI Model Loaded (Offline Mode)"); return true; }

void classify(float* features, char* result, float* confidence) {
    if (!_modelLoaded) {
        strcpy(result, "model_not_loaded");
        *confidence = 0.0;
        return;
    }

    // Normalize input features
    float normalized[4];
    normalized[0] = (features[0] - 20.0) / 20.0; // temp: normalize
around 20°C
    normalized[1] = (features[1] - 50.0) / 50.0; // humidity:
normalize around 50%
    normalized[2] = log10(features[2]) / 6.0; // gas: log scale
    normalized[3] = features[3] / 500.0; // IAQ: 0-500
scale

    // Forward pass - Hidden Layer
    float hidden[8];
    for (int i = 0; i < 8; i++) {

```

```

        hidden[i] = _biasH[i];
        for (int j = 0; j < 4; j++) {
            hidden[i] += normalized[j] * _weightsIH[j][i];
        }
        hidden[i] = relu(hidden[i]);
    }

    // Forward pass - Output Layer
    float output[NUM_CATEGORIES];
    float maxOutput = -1000.0;
    int maxIndex = 0;

    for (int i = 0; i < NUM_CATEGORIES; i++) {
        output[i] = _biasO[i];
        for (int j = 0; j < 8; j++) {
            output[i] += hidden[j] * _weightsHO[j][i];
        }
        if (output[i] > maxOutput) {
            maxOutput = output[i];
            maxIndex = i;
        }
    }

    // Softmax for confidence
    float sum = 0.0;
    for (int i = 0; i < NUM_CATEGORIES; i++) {
        output[i] = exp(output[i] - maxOutput);
        sum += output[i];
    }

    *confidence = output[maxIndex] / sum;
    strcpy(result, SMELL_CATEGORIES[maxIndex]);
}

void trainOnline(float* features, const char* label) {
    // Online learning capability for adaptation
    // This allows the device to learn new smell patterns
    int labelIndex = -1;
    for (int i = 0; i < NUM_CATEGORIES; i++) {
        if (strcmp(label, SMELL_CATEGORIES[i]) == 0) {
            labelIndex = i;
            break;
        }
    }

    if (labelIndex >= 0) {
        // Simple weight update (gradient descent)
        float learningRate = 0.01;
        // ... training logic
        Serial.printf("Model adapted for: %s\n", label);
    }
}

private: bool _modelLoaded = false;

// Network weights
float _weightsIH[4][8]; // Input to Hidden
float _biasH[8]; // Hidden bias
float _weightsHO[8][10]; // Hidden to Output
float _biasO[10]; // Output bias

```

```

float relu(float x) {
    return x > 0 ? x : 0;
}

void initializeWeights() {
    // Pre-trained weights embedded in firmware
    // These would be trained offline and embedded here

    // Input to Hidden weights
    float ih[4][8] = {
        {0.5, -0.3, 0.8, -0.2, 0.4, 0.1, -0.5, 0.3},
        {-0.2, 0.6, -0.1, 0.4, -0.3, 0.5, 0.2, -0.4},
        {0.7, 0.2, -0.4, 0.6, 0.1, -0.3, 0.8, 0.0},
        {-0.1, 0.4, 0.3, -0.5, 0.6, 0.2, -0.2, 0.5}
    };
    memcpy(_weightsIH, ih, sizeof(ih));

    // Hidden biases
    float bh[8] = {0.1, -0.1, 0.2, 0.0, -0.2, 0.1, 0.0, -0.1};
    memcpy(_biasH, bh, sizeof(bh));

    // Hidden to Output weights (simplified)
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 10; j++) {
            _weightsH0[i][j] = ((float)random(-100, 100)) / 100.0;
        }
    }

    // Output biases
    for (int i = 0; i < 10; i++) {
        _bias0[i] = 0.0;
    }
}

};

#endif // SMELL_CLASSIFIER_H

=====
===== 4.
BACKEND API (bio-sentinel.ts)
=====
=====

/** * XBio Sentinel Backend API * Handles device data, storage, and
analysis */

import { Router } from 'express'; import { db } from '../db'; import {
sensorReadings, smellProfiles, devices } from '../shared/schema';
import { eq, desc, and, gte } from 'drizzle-orm';

const router = Router();

// Types interface SensorReading { deviceId: string; temperature:
number; humidity: number; pressure: number; gasResistance:
number; iaq: number; voc: number; eCO2: number; smellClass: string;
confidence: number; timestamp: Date; }

// POST /api/xbio/reading - Store new sensor reading
router.post('/reading', async (req, res) => { try { const reading:
SensorReading = req.body;

```

```

// Validate device
const device = await db.query.devices.findFirst({
  where: eq(devices.id, reading.deviceId)
});

if (!device) {
  return res.status(404).json({ error: 'Device not found' });
}

// Store reading
const result = await db.insert(sensorReadings).values({
  deviceId: reading.deviceId,
  temperature: reading.temperature,
  humidity: reading.humidity,
  pressure: reading.pressure,
  gasResistance: reading.gasResistance,
  iaq: reading.iaq,
  voc: reading.voc,
  eCO2: reading.eCO2,
  smellClass: reading.smellClass,
  confidence: reading.confidence,
  timestamp: new Date()
}).returning();

// Update device last seen
await db.update(devices)
  .set({ lastSeen: new Date() })
  .where(eq(devices.id, reading.deviceId));

// Check for alerts
await checkAlerts(reading);

res.json({ success: true, id: result[0].id });

} catch (error) { console.error('Error storing reading:', error);
res.status(500).json({ error: 'Failed to store reading' }); } });

// GET /api/xbio/readings/:deviceId - Get readings for device
router.get('/readings/:deviceId', async (req, res) => { try { const {
deviceId } = req.params; const { hours = 24 } = req.query;

const since = new Date();
since.setHours(since.getHours() - Number(hours));

const readings = await db.query.sensorReadings.findMany({
  where: and(
    eq(sensorReadings.deviceId, deviceId),
    gte(sensorReadings.timestamp, since)
  ),
  orderBy: desc(sensorReadings.timestamp),
  limit: 1000
});

res.json(readings);

} catch (error) { console.error('Error fetching readings:', error);
res.status(500).json({ error: 'Failed to fetch readings' }); } });

// GET /api/xbio/analysis/:deviceId - Get AI analysis
router.get('/analysis/:deviceId', async (req, res) => { try { const {
deviceId } = req.params;

```

```

// Get last 100 readings
const readings = await db.query.sensorReadings.findMany({
  where: eq(sensorReadings.deviceId, deviceId),
  orderBy: desc(sensorReadings.timestamp),
  limit: 100
});

if (readings.length === 0) {
  return res.json({ status: 'no_data' });
}

// Calculate statistics
const analysis = {
  avgTemperature: average(readings.map(r => r.temperature)),
  avgHumidity: average(readings.map(r => r.humidity)),
  avgIAQ: average(readings.map(r => r.iaq)),
  avgVOC: average(readings.map(r => r.voc)),
  avgEC02: average(readings.map(r => r.ec02)),
  dominantSmell: findDominant(readings.map(r => r.smellClass)),
  airQualityStatus: getAirQualityStatus(average(readings.map(r =>
r.iaq))),
  trend: calculateTrend(readings),
  alerts: generateAlerts(readings),
  lastReading: readings[0],
  periodHours: 24,
  readingCount: readings.length
};

res.json(analysis);

} catch (error) { console.error('Error generating analysis:', error);
res.status(500).json({ error: 'Failed to generate analysis' }); } });

// POST /api/xbio/profile - Create smell profile router.post('/profile',
async (req, res) => { try { const { name, description,
baselineReadings, deviceId, userId } = req.body;

// Calculate baseline values from readings
const baseline = {
  temperature: average(baselineReadings.map((r: any) =>
r.temperature)),
  humidity: average(baselineReadings.map((r: any) => r.humidity)),
  gasResistance: average(baselineReadings.map((r: any) =>
r.gasResistance)),
  iaq: average(baselineReadings.map((r: any) => r.iaq))
};

const profile = await db.insert(smellProfiles).values({
  name,
  description,
  baseline: JSON.stringify(baseline),
  deviceId,
  userId,
  createdAt: new Date()
}).returning();

res.json({ success: true, profile: profile[0] });

} catch (error) { console.error('Error creating profile:', error);
res.status(500).json({ error: 'Failed to create profile' }); } });

```

```

// Helper functions
function average(arr: number[]): number { return
arr.reduce((a, b) => a + b, 0) / arr.length; }

function findDominant(arr: string[]): string { const counts:
Record<string, number> = {}; arr.forEach(item => { counts[item] =
(counts[item] || 0) + 1; }); return Object.entries(counts).sort((a, b) =>
b[1] - a[1])[0][0]; }

function getAirQualityStatus(iaq: number): string { if (iaq <= 50)
return 'excellent'; if (iaq <= 100) return 'good'; if (iaq <= 150) return
'moderate'; if (iaq <= 200) return 'poor'; if (iaq <= 300) return
'unhealthy'; return 'hazardous'; }

function calculateTrend(readings: any[]): string { if (readings.length <
10) return 'insufficient_data';

const recent = readings.slice(0, 10); const older = readings.slice(-10);

const recentAvg = average(recent.map(r => r.iaq)); const olderAvg =
average(older.map(r => r.iaq));

const diff = recentAvg - olderAvg; if (diff > 20) return 'improving'; if
(diff < -20) return 'declining'; return 'stable'; }

function generateAlerts(readings: any[]): string[] { const alerts:
string[] = []; const latest = readings[0];

if (latest.iaq > 200) alerts.push('Poor air quality detected'); if
(latest.voc > 1.0) alerts.push('High VOC levels'); if (latest.eCO2 >
1000) alerts.push('High CO2 levels'); if (latest.humidity > 70)
alerts.push('High humidity'); if (latest.humidity < 30) alerts.push('Low
humidity');

return alerts; }

async function checkAlerts(reading: SensorReading) { // Real-time
alert checking and notification if (reading.iaq > 300) {
console.log(ALERT: Hazardous air quality on device
`${reading.deviceId}`); // Send push notification, email, etc. } }

export default router;

=====
===== 5.
REACT DASHBOARD (XBioSentinel.tsx)
=====
=====

/** * XBio Sentinel Dashboard Component * Real-time environmental
monitoring interface */

import React, { useState, useEffect } from 'react'; import { useQuery
} from '@tanstack/react-query'; import { Card, CardHeader,
CardTitle, CardContent } from '@components/ui/card'; import {
Badge } from '@components/ui/badge'; import { Thermometer,
Droplets, Wind, Activity, AlertTriangle, CheckCircle, TrendingUp,
TrendingDown } from 'lucide-react';

interface SensorReading { id: string; temperature: number; humidity:
number; pressure: number; gasResistance: number; iaq: number; voc:
number; eCO2: number; smellClass: string; confidence: number;
timestamp: string; }

```

[illegible]

```

        <"Card className="bg-red-900/20 border-red-500>
            <"CardContent className="p-4>
                <"div className="flex items-center gap-2 text-red-400>
                    </ "AlertTriangle className="h-5 w-5>
                    <span/><span className="font-bold>تنبيهات:</span>
                <div/>
                    <"ul className="mt-2 space-y-1>
                        ) <= (analysis.alerts.map((alert, i)
                            li key={i} className="text-red-300 text-sm">• {alert}>
                    <</li>
                <ul/>
            <CardContent/>
        <Card/>
    }

    /* Main Stats Grid */
    <"div className="grid grid-cols-2 md:grid-cols-4 gap-4>
        /* Temperature */
        <"Card className="bg-gray-800 border-gray-700>
            <"CardContent className="p-4>
                div className="flex items-center gap-2 text-orange-400 mb->
                    </ "Thermometer className="h-5 w-5>
                    <span/>درجة الحرارة<span className="text-sm>
                <div/>
                <"div className="text-3xl font-bold text-white>
                    lastReading.temperature.toFixed(1)}°C
                <div/>
            <CardContent/>
        <Card/>

        /* Humidity */
        <"Card className="bg-gray-800 border-gray-700>
            <"CardContent className="p-4>
                <"div className="flex items-center gap-2 text-blue-400 mb-2>
                    </ "Droplets className="h-5 w-5>
                    <span/>الرطوبة<span className="text-sm>
                <div/>
                <"div className="text-3xl font-bold text-white>
                    %{lastReading.humidity.toFixed(1)}%
                <div/>
            <CardContent/>
        <Card/>

        /* IAQ */
        <"Card className="bg-gray-800 border-gray-700>
            <"CardContent className="p-4>
                div className="flex items-center gap-2 text-green-400 mb->
                    </ "Wind className="h-5 w-5>
                    <span/>مؤشر جودة الهواء<span className="text-sm>
                <div/>
                <"div className="text-3xl font-bold text-white>
                    {(lastReading.iaq.toFixed(0))}
                <div/>
                <"div className="text-xs text-gray-400 mt-1>
                    {(getIAQDescription(lastReading.iaq))}
                <div/>
            <CardContent/>

```

```

</Card/>

        { /* Smell Classification */ }
        <"Card className="bg-gray-800 border-gray-700>
            <"CardContent className="p-4>
div className="flex items-center gap-2 text-purple-400 mb->
                                                    <"2
                </ "Activity className="h-5 w-5>
                <span/> تصنيف الرائحة <"span className="text-sm>
                    <div/>
                <"div className="text-xl font-bold text-white capitalize>
                    {lastReading.smellClass}
                <div/>
                <"div className="text-xs text-gray-400 mt-1>
                %{{(lastReading.confidence * 100).toFixed(1)}} : ثقة
                <div/>
                <CardContent/>
                <Card/>
                <div/>

                { /* Detailed Stats */ }
                <"div className="grid grid-cols-1 md:grid-cols-3 gap-4>
                    { /* VOC */ }
                    <"Card className="bg-gray-800 border-gray-700>
                        <"CardContent className="p-4>
                        <div className="text-gray-400 text-sm mb-1>
                            <div/>
                        <div className="text-2xl font-bold text-white">>
                            <{{lastReading.voc.toFixed(2)}} ppm</div>
                        <CardContent/>
                        <Card/>

                        { /* eCO2 */ }
                        <"Card className="bg-gray-800 border-gray-700>
                            <"CardContent className="p-4>
                            <div className="text-gray-400 text-sm mb-1>
                                <div/>
                            <div className="text-2xl font-bold text-white">>
                                <{{lastReading.eCO2.toFixed(0)}} ppm</div>
                            <CardContent/>
                            <Card/>

                            { /* Trend */ }
                            <"Card className="bg-gray-800 border-gray-700>
                                <"CardContent className="p-4>
                                <div/> <div/> الاتجاه <"div className="text-gray-400 text-sm mb-1>
                                    <"div className="flex items-center gap-2>
                                        ) ? 'analysis.trend === 'improving'
TrendingUp className="h-6 w-6 text-green-500" /><span><>
                                    </><span/> تحسن <"className="text-green-500 font-bold
                                        ) ? 'analysis.trend === 'declining : (
TrendingDown className="h-6 w-6 text-red-500" /><span><>
                                    </><span/> تراجع <"className="text-red-500 font-bold
                                        ) : (
CheckCircle className="h-6 w-6 text-blue-500" /><span><>
                                    </><span/> مستقر <"className="text-blue-500 font-bold
                                        { (
                                        <div/>
                                    <CardContent/>
                                    <Card/>

```

<div/>
<div/>

{ ;(

=====
نهاية الملف
=====
=====

جميع الحقوق محفوظة (c) 2026 ARC Advanced Environmental Technologies
SA 1020258841 :د المراجعة