ERZİNCAN BİNALİ YILDIRIM ÜNİVERSİTESİ
MÜHENDİSLİK-MİMARLIK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

MİKRODENETLEYİCİ UYGULAMALARI DERSİ

212701005
Habil Yusuf AYHAN

NodeMCU ile Acil Durum Akıllı Ev Projesi

İçindekiler

ŞEKİLLER

1.GİRİŞ

1.1 Projenin Amacı ve Özellikleri

Günümüz teknolojisinde IoT (Nesnelerin İnterneti) uygulamaları, çeşitli sektörlerde yenilikçi çözümler sunarak hem bireysel hem de toplumsal fayda sağlamaktadır. Bu bağlamda geliştirilen projemiz, **NodeMCU**, **MQ-2 gaz sensörü, DHT11 sıcaklık sensörü** ve **ADXL345 jiroskop sensörü** gibi temel elektronik bileşenleri kullanarak çevresel verilerin izlenmesi ve anormal durumlara hızlı tepki verilmesi üzerine odaklanmaktadır. Proje, gerçek zamanlı veri ölçümü ve analizi yaparak hem donanımsal hem de yazılımsal düzeyde kullanıcıya bilgilendirme sağlar.

1.1.1 Projenin Amacı

Bu proje, çevresel güvenliği artırmak ve kullanıcılara hızlı geri bildirim sunmak için bir sistem geliştirmeyi amaçlamaktadır. Sistemin temel amacı; gaz yoğunluğu, sıcaklık ve sismik hareketleri (deprem belirtileri) sürekli izleyerek:

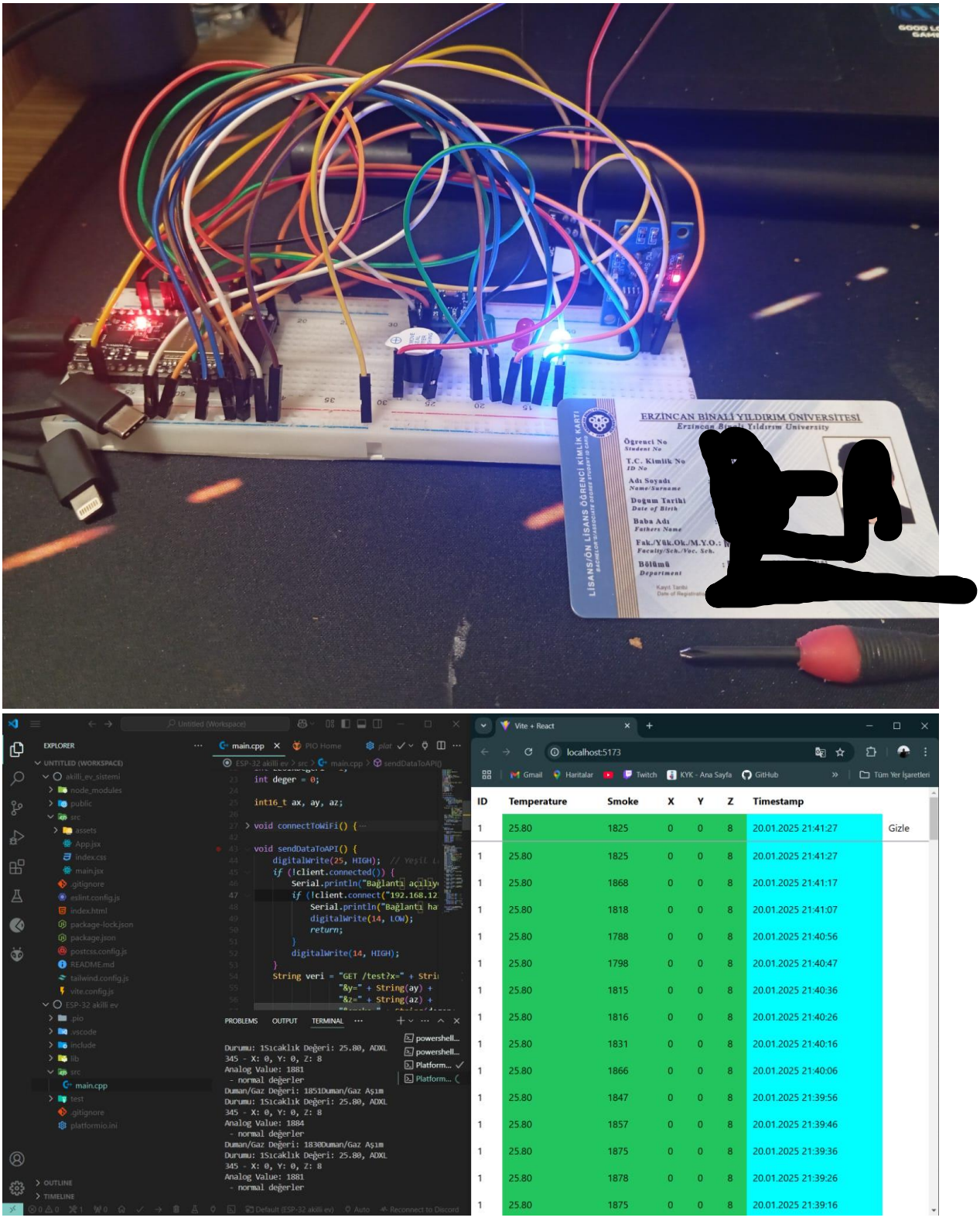
- **Tehlikeli gaz seviyelerini tespit etmek,**
- **Sıcaklık değerlerini ölçmek ve raporlamak,**
- **Olası deprem hareketlerini algılayarak uyarılar göndermek** üzerine odaklanır.

Bu verilere dayalı olarak, anormal bir durum tespit edildiğinde hem donanım üzerinden görsel ve işitsel uyarılar sağlanacak, hem de web tabanlı bir arayüz aracılığıyla kullanıcılar bilgilendirilecektir.

1.1.2 Proje Özellikleri

1. **Donanım Modülleri:**
    - **MQ-2 Gaz Sensörü:** Ortamdaki yanıcı gazlar (LPG, metan, duman vb.) ve havadaki gaz yoğunluğunu ölçer.
    - **ADXL345 Jiroskop Sensörü:** Deprem benzeri hareketleri algılamak için ölçümler yapar.
    - **DHT11 Sıcaklık Sensörü:** Olağan dışı sıcaklık değerlerini algılamak için ölçümler yapar.
    - **NodeMCU (ESP32):** Verilerin işlenmesi, saklanması ve API aracılığıyla web platformuna gönderilmesinden sorumlu olan merkezi mikrodenetleyici.
2. **Web Tabanlı İzleme ve Kontrol:**
    NodeMCU tarafından toplanan veriler, bir **RESTful API** üzerinden sunucuya gönderilir. Kullanıcılar, bu verileri gerçek zamanlı olarak bir web sitesi üzerinden takip edebilir. Web sitesinde, gaz yoğunluğu, sıcaklık ve sismik hareketlere ilişkin grafikler ve raporlar görüntülenebilir.
3. **Anormal Durumlarda Alınacak Aksiyonlar:**
    Gaz seviyesi veya sıcaklık belirlenen eşik değeri aşarsa veya sismik hareket şiddeti yüksek bir seviyeye ulaşırsa:
        - Donanım üzerinde **LED'ler yanar** ve **buzzer alarm verir.**
        - Web sitesinde kullanıcıya gönderilen değer **kırmızı arka plan renginde** gösterilir.
        - Gerekirse e-posta veya SMS gibi alternatif yollarla da kullanıcı bilgilendirilebilir.

2.PROJE FOTOĞRAFLARI



Şekil 1: Proje Fotoğrafları

3.PROJE KODLARININ AÇIKLANMASI

3.1. Gerekli Kütüphanelerin Dahil Edilmesi

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_ADXL345_U.h>
#include <WiFi.h>
#include <DHT.h>
```

Şekil 2: Kütüphaneler

Bu bölümde kullanılan sensörler ve cihazlar için gerekli kütüphaneler dahil edilir:

- **Adafruit_ADXL345_U.h**: ADXL345 sensöründen veri okumak için.
- **Wire.h**: I2C protokolüyle haberleşme için.
- **WiFi.h**: NodeMCU'nun Wi-Fi bağlantısını sağlamak için.

3.2. Wi-Fi Bağlantısı İçin Ayarların Yapılması

```
const char* agAdi = "redmi12";
const char* agSifresi [redacted]
```

Şekil 3: Wifi Ayarları

- **agAdi**: Wi-Fi ağınızın SSID'si.
- **agSifresi**: Wi-Fi ağınızın şifresi.

Bu bilgiler NodeMCU'nun Wi-Fi ağına bağlanmasını sağlar.

3.3. Donanım ve Sensör Değişkenleri

```
float temperature = 0.0;
int currentX = 0;
int currentY = 0;
int currentZ = 0;
int gasStatus = 0;
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);
WiFiClient client;
#define DHTPIN 15
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
int esikDegeri = 700;
int xEsikDegeri = 1;
int yEsikDegeri = 1;
int zEsikDegeri = 1;
int deger = 0;
```

Şekil 4: Değişkenler

- **currentX, currentY, currentZ**: ADXL345'in önceki konum değerlerini saklar.
- **esikDegeri**: Gaz sensöründen gelen verinin eşik değeri.

- **xEsikDegeri, yEsikDegeri, zEsikDegeri**: Hareket algılamada kullanılacak eksen eşik değerleri.
- **temperature**: DHT11 sıcaklık sensöründen okunan sıcaklık verisini tutan değişken.

3.4. Wi-Fi Bağlantısını Sağlama

```
void connectToWiFi() {
    Serial.println("WiFi'ye bağlanıyor...");

    WiFi.begin(agAdi, agSifresi);

    while (WiFi.status() != WL_CONNECTED) {

        delay(1000);
        Serial.println("Bağlanıyor...");
        digitalWrite(14, LOW);
    }
    Serial.println("Bağlandı! IP adresi: ");
    digitalWrite(14, HIGH);
    Serial.println(WiFi.localIP());
}
```

Şekil 5: Wifi Bağlantı Kodları

- **WiFi.begin()** ile NodeMCU'yu Wi-Fi ağına bağlar.
- Bağlantı başarılı olduğunda cihazın IP adresini seri port üzerinden yazdırır.

3.5. Verileri API'ye Gönderme

```
void sendDataToAPI() {
    digitalWrite(25, HIGH);  // Yeşil LED
    if (!client.connected()) {
        Serial.println("Bağlantı açılıyor...");
        if (!client.connect("192.168.131.46", 3000)) {
            Serial.println("Bağlantı hatası!");
            digitalWrite(14, LOW);
            return;
        }
        digitalWrite(14, HIGH);
    }
    String veri = "GET /test?x=" + String(ax) +
                  "&y=" + String(ay) +
                  "&z=" + String(az) +
                  "&smoke=" + String(deger) +
                  "&temperature=" + String(temperature) +
                  "&gasstatus=" + String(gasStatus) +
                  "&id=1 HTTP/1.1\r\n";
    veri += "Host: 195.85.216.225\r\n";
    veri += "Connection: close\r\n\r\n";
    client.print(veri);
    Serial.println("Veri gönderildi");

    while (client.available()) {
        String response = client.readString();
        Serial.println(response);
    }
    client.stop();
    digitalWrite(25, LOW);  // Yeşil LED'i kapat
}
```

Şekil 6: Verilerin Okunması

Bu fonksiyon:

- API'ye veri göndermek için **HTTP GET** isteği yapar.
- Sıcaklık, gaz, hareket değerlerini ve sıcaklık sensörünün dijital pininden gelen "gasstatus" değerini bir URL üzerinden gönderir.
- Veri gönderildikten sonra gelen response verisini seri porta yazdırır.
- Veri göndermeye başladığı esnada yeşil ledi yakar veri gönderimi tamamlandığında yeşil ledi söndürür.

3.6. Donanımın Kurulması

```
void setup() {
    Wire.begin(21, 22);
    Serial.begin(9600);

    if (!accel.begin()) {
        Serial.println("ADXL345 bulunamadı! Kontrol edin.");
        while (1);
    }
    Serial.println("ADXL345 bulundu!");
    pinMode(26, OUTPUT); // Kırmızı LED
    pinMode(27, OUTPUT); // Buzzer
    pinMode(25, OUTPUT); // Yeşil LED
    pinMode(34, INPUT);  // Duman/Gaz sensörü (Analog giriş)
    pinMode(14, OUTPUT);  // Mavi Led (Dijital giriş)
    pinMode(13, INPUT);
    connectToWiFi();
}
```

Şekil 7: Setup Kodları

Bu fonksiyon seri haberleşmenin bant hızını 9600 olarak ayarlar. Ardından ADXL345 sensöründen veri gelip gelmediğinin kontrolünü yapar. ADXL345 sensörünün başlangıç frekans değerleri ve hassasiyeti ayarlanır. 26, 27, 25 ve 14 pinleri sırasıyla kırmızı led, buzzer, yeşil led ve mavi ledi temsil eder. Pin 34 Duman sensöründen "INPUT" alınacağını belirterek tanımlanır. En son olarak "connectToWiFi" fonksiyonu çağrılarak WiFi a bağlanılır.

3.7 Sensörlerden Veri Okunması, API ye Veri Gönderilmesi ve Alarm Durumlarının Kontrolü

```
void loop() {
    sensors_event_t event;
    accel.getEvent(&event);
    temperature = dht.readTemperature();
    ax = event.acceleration.x;
    ay = event.acceleration.y;
    az = event.acceleration.z;
    gasStatus = digitalRead(13);
    deger = analogRead(34);
    deger = map(deger, 0, 1023, 0, 1000);
    Serial.print("Duman/Gaz Değeri: ");
    Serial.print(deger);
    Serial.print("Duman/Gaz Aşım Durumu: ");
    Serial.print(gasStatus);
    Serial.print("Sıcaklık Değeri: ");
    Serial.print(temperature);
    Serial.print(", ADXL345 - X: ");
    Serial.print(ax);
    Serial.print(", Y: ");
    Serial.print(ay);
    Serial.print(", Z: ");
    Serial.println(az);
    int analogValue = analogRead(34);
    Serial.print("Analog Value: ");
    Serial.println(analogValue);
    bool alarm = false;
    if (temperature > 29){
        Serial.println(" - yüksek Sıcaklık");
        alarm = true;
    }
    if (gasStatus == 0) {
        Serial.println(" - yüksek duman veya gaz");
        alarm = true;
    }
```

```
if (abs(ax - currentX) > xEsikDegeri) {
    Serial.println(" - X ekseni için eşik değeri aşıldı");
    alarm = true;
}
if (abs(ay - currentY) > yEsikDegeri) {
    Serial.println(" - Y ekseni için eşik değeri aşıldı");
    alarm = true;
}
if (abs(az - currentZ) > zEsikDegeri) {
    Serial.println(" - Z ekseni için eşik değeri aşıldı");
    alarm = true;
}
currentX = ax;
currentY = ay;
currentZ = az;
if (alarm) {
    digitalWrite(26, HIGH);  // Kırmızı LED
    digitalWrite(27, HIGH);  // Buzzer
    digitalWrite(25, LOW);   // Yeşil LED
    delay(100);
    digitalWrite(26, LOW);   // Kırmızı LED kapalı
    digitalWrite(27, LOW);   // Buzzer kapalı
} else {
    Serial.println(" - normal değerler");
    digitalWrite(26, LOW);   // Kırmızı LED kapalı
    digitalWrite(27, LOW);   // Buzzer kapalı
    digitalWrite(25, LOW);   // Yeşil LED kapalı
}

static unsigned long lastSendTime = 0;
if (millis() - lastSendTime >= 10000) {
    sendDataToAPI();
    lastSendTime = millis();
}
delay(100);
```

Şekil 8: Loop Kodları

ADXL345 jiroskop sensörü, DHT11 sıcaklık sensörü ve MQ-2 gaz sensöründen okunan veriler her 10 saniyede bir API ye gönderilir. Verilerde olağandışı bir durum olduğunda, kırmızı led ve buzzerlar çalışarak bu durumu kullanıcıya bildirirler.

4.API KODLARI

4.1 API nin Main Dosyasının Kodlanması

```
const dotenv = require("dotenv").config();
const express = require("express");
const { Server } = require("socket.io");
const cors = require("cors");
const app = express();
require("./src/config/database");
const authRouter = require("./src/routers/auth_router");
app.use(cors());
app.use(express.urlencoded({ limit: "10mb", extended: true }));
app.use(express.json({ limit: "10mb" }));
app.use("/", authRouter);
const PORT = process.env.PORT || 3000;
const server = app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
const io = new Server(server, {
  cors: {
    origin: "*", // Gerekirse buraya frontend URL'nizi ekleyin
    methods: ["GET", "POST"],
  },
});
io.on("connection", (socket) => {
  console.log(`New client connected: ${socket.id}`);
  socket.on("disconnect", () => {
    console.log(`Client disconnected: ${socket.id}`);
  });
});
module.exports = { io };
```

Şekil 9: App.js Kodları

"Dotenv" modülü kullanılarak proje dizinindeki kod üzerinde açık bir şekilde göstermek istemediğimiz verilerin tutulduğu ".env" dosyasına erişim sağlandı. "Express" modülü ile kurulan apinin anlık veri gönderebilmesi için "socket.io" modülünün kullanılması gerekti. "cors" modülü proje URL si içerisindeki tüm portlardan gelen istekleri yakalamak için kullanıldı. Bunun sebebi projenin frontend kısmı olan react tarafında projenin 5173 portunda çalışıyor olmasıdır.

4.2 Controller dosyası

```
const Data = require("../model/data_model");
const saveData = async (req, res, next) => {
  const { io } = require("../../app");
  console.log(io);
  try {
    console.log(req.query);
    id = Number(req.query.id);
    let data = await Data.findOne({ id: id });
    if (!data) {
      data = new Data({ id });
    }
    data.x.push(req.query.x);
    data.y.push(req.query.y);
    data.z.push(req.query.z);
    data.gasStatus.push(req.query.gasstatus);
    data.temperature.push(req.query.temperature);
    data.smoke.push(req.query.smoke);
    data.times.push(Date.now());
    await data.save();
    io.emit("data", data);
    res.status(200).json({ message: "Data saved and emitted" });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "An error occurred" });
  }
};
const getAllData = async (req, res, next) => {
  console.log("data istedi");
  let data = await Data.find();
  return res.json(data);
};
module.exports = {
  saveData,
  getAllData,
};
```

Şekil 10: Controller Kodları

Controller dosyasında bulunan "saveData" fonksiyonu donanımdan gelen verileri veritabanına kaydetmekle sorumludur. "getAllData" fonksiyonu ise veritabanındaki tüm verileri çekip response olarak göndermekten sorumludur.

4.3 Rooter'ların tanımlanması

```
const router = require("express").Router();
const authController = require("../controllers/auth_controller");

router.get("/test", authController.saveData);
router.get("/getalldata", authController.getAllData);

module.exports = router;
```

Şekil 11: Rooter Kodları

5.SONUÇ

Bu proje, IoT tabanlı akıllı ev sistemlerinin temelini oluşturan bir çözüm sunarak, çevresel güvenlik ve kullanıcı bilgilendirme süreçlerini başarılı bir şekilde bir araya getirmiştir. NodeMCU, MQ-2 gaz sensörü, DHT11 sıcaklık sensörü ve ADXL345 jiroskop sensörü gibi bileşenlerin etkin bir şekilde entegre edildiği sistem, hem donanımsal hem de yazılımsal düzeyde güvenilir ve esnek bir yapı sergilemektedir.
Projenin en önemli katkılarından biri, çevredeki anormal durumların (örneğin gaz kaçağı, yüksek sıcaklık ya da sismik hareketler) tespiti için uygun maliyetli, taşınabilir ve kullanıcı dostu bir çözüm sunmasıdır. Bu tür sistemlerin günlük yaşantıda artan kullanımı, bireylerin ve toplulukların güvenliğini artırmaya yönelik önemli bir adım olarak değerlendirilebilir.

6.KAYNAKÇA

- https://learn.adafruit.com/adxl345-digital-accelerometer
- https://www.espressif.com/en/products/socs/esp32/resources
- https://www.arduino.cc/reference/en/libraries/wifi/
- https://learn.adafruit.com/dht