

ÜNİVERSİTE ÖĞRENCİLERİ ARAŞTIRMA PROJELERİ DESTEK PROGRAMI
SONUÇ RAPORU

PROJE BAŞLIĞI: Mobil Hava Kalitesi İzleme Sistemi Tasarımı ve Uygulaması

PROJE YÜRÜTÜCÜSÜNÜN ADI: Erhan ÖZDOĞAN

DANIŞMANININ ADI: Dr. Öğr. Üyesi Mehmet Kürşat YALÇIN

GENEL BİLGİLER

PROJENİN KONUSU	Günümüzde hava kirliliği, sağlık sorunlarına ve çevresel sorunlara yol açabilen önemli bir sorundur. Hava kalitesi izleme sistemleri, bu sorunu izlemek, analiz etmek ve kontrol etmek için önemli bir araçtır. Bu proje, taşınabilir bir hava kalitesi izleme sistemi tasarlayarak ve uygulayarak hava kirliliğinin izlenmesi ve analiz edilmesine katkıda bulunmayı amaçlamaktadır.
PROJE YÜRÜTÜCÜSÜNÜN ADI	Erhan ÖZDOĞAN
DANIŞMANIN ADI	Dr. Öğr. Üyesi Mehmet Kürşat YALÇIN
PROJE BAŞLANGIÇ VE BİTİŞ TARİHLERİ	22.03.2024 – 26.06.2024

1.Giriş

Bu projenin temel amacı, taşınabilir bir hava kalitesi izleme sistemi tasarlamak, geliştirmek ve hava kirliliğinin anlık ve sürekli olarak izlenmesine olanak tanıyan bir çözüm sunmaktır. Mobil hava kalitesi izleme sistemi, çeşitli hava kirliliği parametrelerini ölçebilme yeteneği, taşınabilirliği ve kullanıcı dostu bir arayüze sahip olma özellikleriyle ön plana çıkmaktadır.

Çeşitli Parametreleri İzleme: Projede, atmosferdeki önemli hava kirliliği parametrelerini ölçebilecek şekilde sensör seçimi yapılmıştır.

Taşınabilirlik: Mobil hava kalitesi izleme cihazı, kullanıcının kolaylıkla taşıyabileceği ve farklı bölgelerde hava kalitesini izleyebileceği bir form faktörüne sahiptir. Bu, cihazın çeşitli ortamlarda kullanılabilmesini sağlamıştır.

Veri Toplama ve Depolama: Proje, sensörlerden gelen verilerin güvenli bir şekilde toplanmasını ve depolanmasını sağlayacak bir SD kart modülü içermektedir. Bu sistem, uzun vadeli veri analizleri için güçlü bir temel sağlamaktadır.

Veri Görselleştirme: Toplanan veriler kullanıcı dostu bir arayüzde görselleştirilmiştir. Bu, hava kalitesi değişimlerini anlamak ve sonuçları anlamlı bir şekilde yorumlamak için kullanıcılara olanak tanır.

Raporlama ve Bilinç Oluşturma: Proje tamamlanmıştır ve elde edilen sonuçları ve bulguları içeren bu rapor hazırlanmıştır. Aynı zamanda, hava kirliliği ve hava kalitesi izleme konularında toplumda farkındalık oluşturmak amacıyla bilgi paylaşımı yapılacaktır.

2. Rapor Döneminde Yapılan Çalışmalar

2.1 Donanım alıřmaları

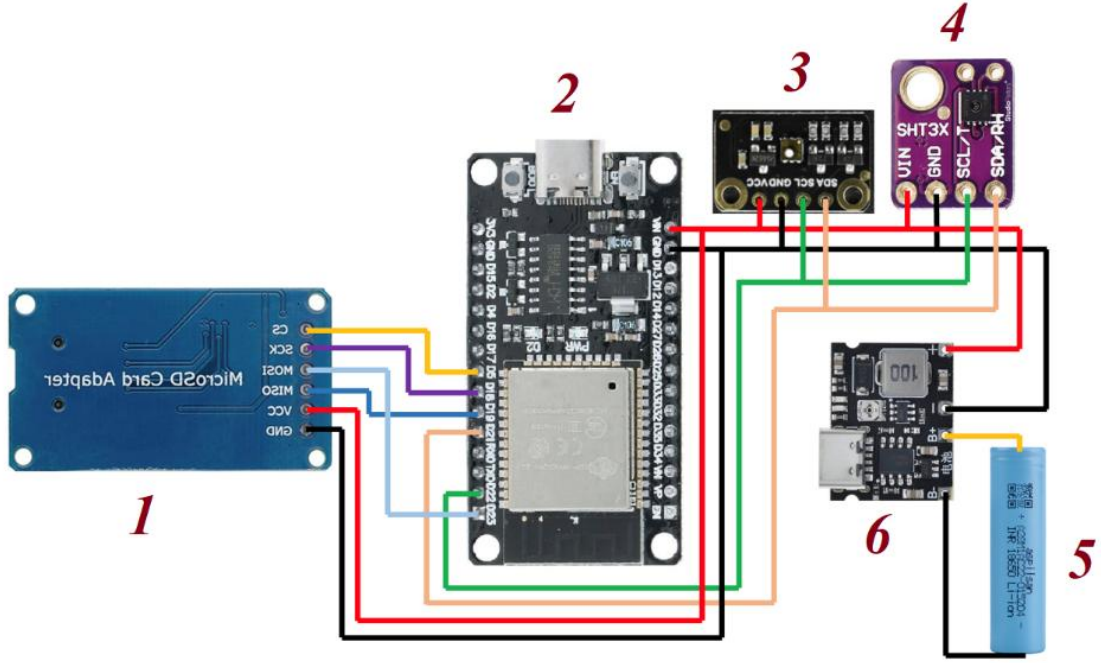
Projede ilk olarak sensör seimleri ve takiben temini yapılmıřtır. Bu sensörler řunlardır. Hava kalitesi için SGP40 (řekil 1-3) ve bu sensörün doėruluėunu arttırmak üzere sıcaklık ve nem sensörü olarak SHT31 (řekil 1-4) seilmiřtir.

Bu sensörleri hızlı test etmek üzere wi-fi özelliėi bulunan ESP32 modülü de (řekil 1-2) satın alınmıřtır. Sistemi daha yetenekli kılmak ve olası veri işlemlerini gömülü gerçekleştirilebilirliėini test etmek amacı ile proje kapsamında Raspberry Pi modüller de satın alınmıřtır. Proje takvimi doėrultusunda kullanımları deėerlendirilecektir.

Sistem mobil olacaėından tek hücre bir Li-Ion pille (řekil 1-5) sistemi beslemeye karar verilmiřtir. Gerek ESP32 gerekse sensörlerin alıřma gerilim aralıkları incelendiėinde lityum pil gerilim aralıėının (2.7V - 4.2V) uygun olmadığı dolayısıyla bir dc/dc gerilim yükseltici ve düzenleyici (boost regülatör) ihtiyacı olduėu deėerlendirilmiřtir. Ayrıca pil bittiėinde řarj etmek üzere bir řarj kontrolcüsüne ihtiyaç vardır. Bu iki fonksiyonu bir arada saėlayan bir modül (řekil 1-6) bulunmuř ve satın alınmıřtır.

Alınan verilerin cihaz ierisinde de saklanması adına bir hafıza kartına depolanması kararlařtırılmıřtır. Bunun için mikro sd kart ve ESP32 modülüne bu kartı baėlayabilmek için bir kart adaptörü (řekil 1-1) alınmıřtır.

Yapılan alıřmanın kavramsal devre řeması řekil 1.'de verilmiřtir.



Şekil 1. Kavramsal Devre Şeması.

2.2 Yazılım çalışmaları

Nihai yazılıma ulaşmak adına sistem bileşenleri bireysel olarak test edilmiştir. Bu amaçla test kodları yazılmıştır.

Test1: İlk test hava kalite sensörü ve nem/sıcaklık sensöründen veri alma ve seri monitörde gösterme testidir. Bu kod, SHT31 ve SGP40 sensörlerini kullanarak sıcaklık, nem ve VOC (Volatile Organic Compounds - Uçucu Organik Bileşikler) indeksini ölçen ve bu değerleri seri port üzerinden yazdıran bir programdır. Her saniyede bir SHT31 sensöründen sıcaklık ve nem değerlerini, SGP40 sensöründen ise VOC indeks değerini alır ve seri port üzerinden bu değerleri yazdırır.

Aşağıda kodlar verilmiştir.

```

#include <DFRobot_SGP40.h>
#include "Wire.h"
#include "SHT31.h"
#define SHT31_ADDRESS 0x44

SHT31 sht;

DFRobot_SGP40 mySGP40; //Declare SGP40.

void setup() {
    Serial.begin(115200);

    while (mySGP40.begin(/*duration = */ 10000) != true) {
        Serial.println("Failed to initialize SGP40, please check wiring and cable
connections"); //You need to press the Gravity cable all the way in.
        delay(1000);
    }

    sht.begin();
    sht.read();

    mySGP40.setRhT(/*relativeHumidity = */ sht.getHumidity(), /*temperature = */
sht.getTemperature());
}

void loop() {
    sht.read();

    Serial.print(sht.getTemperature(), 1);
    Serial.print("\t");
    Serial.print(sht.getHumidity(), 1);
    Serial.print("\t");

    uint16_t index = mySGP40.getVocIndex(); //Declare variable that is to be used
to store VOC index.

    Serial.print("vocIndex = ");
    Serial.println(index);
    delay(1000);
}

```

Test2: Daha sonra SD kart ve modülü için aşağıdaki test kodu yazılmıştır. Bu kod parçası, bir SD kartı kullanarak dosya yazma ve dosyaya veri ekleme işlemlerini gerçekleştirir. SPI iletişimi aracılığıyla SD kart ile etkileşim kurar. Her iki fonksiyon da

dosya işlemlerinin başarılı olup olmadığını kontrol eder ve sonuçları seri port üzerinden bildirir. Bu sayede programın çalışması sırasında dosya işlemleri hakkında bilgi sahibi olunabilir.

```

#include "FS.h"
#include "SD.h"
#include "SPI.h"

void writeFile(fs::FS &fs, const char *path, const char *message) {
    Serial.printf("Writing file: %s\n", path);
    File file = fs.open(path, FILE_WRITE);
    if (!file) {
        Serial.println("Failed to open file for writing");
        return;
    }
    if (file.print(message)) {
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
    file.close();
}

void appendFile(fs::FS &fs, const char *path, const char *message) {
    Serial.printf("Appending to file: %s\n", path);
    File file = fs.open(path, FILE_APPEND);
    if (!file) {
        Serial.println("Failed to open file for appending");
        return;
    }
    if (file.print(message)) {
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}

```

```

void setup() {
    Serial.begin(115200);
    if (!SD.begin()) {
        Serial.println("Card Mount Failed");
        return;
    }
    uint8_t cardType = SD.cardType();

    if (cardType == CARD_NONE) {
        Serial.println("No SD card attached");
        return;
    }

    writeFile(SD, "/data.txt", "Veriler\n");
}

int counter = 0;

void loop() {
    delay(10000);
    String dummy = String(counter++) + "\n";
    appendFile(SD, "/data.txt", dummy.c_str());
}

```

Test3: Bu testte Test1 ve Test2 kodları birleştirilmiştir. Sensörleri 10 saniyede bir okur ve SD karta verileri kaydeder.

```

#include "FS.h"
#include "SD.h"
#include "SPI.h"
#include <DFRobot_SGP40.h>
#include "Wire.h"
#include "SHT31.h"
#define SHT31_ADDRESS 0x44
SHT31 sht;
DFRobot_SGP40 mySGP40; //Declare SGP40.
void setup() {
    Serial.begin(115200);
    while (mySGP40.begin(/*duration = */ 10000) != true) {
        Serial.println("Failed to initialize SGP40, please check wiring
and cable connections"); //You need to press the Gravity cable all
        delay(1000);
    }
    sht.begin();
    sht.read();
    mySGP40.setRhT(/*relativeHumidity = */ sht.getHumidity(),
/*temperature = */ sht.getTemperature());
    if (!SD.begin()) {
        Serial.println("Card Mount Failed");
        return;
    }
    uint8_t cardType = SD.cardType();
    if (cardType == CARD_NONE) {
        Serial.println("No SD card attached");
        return;
    }
    writeFile(SD, "/data.txt", "Veriler\n");
}

```

```
void loop() {
    sht.read();
    float temperature = sht.getTemperature();
    float humidity = sht.getHumidity();

    Serial.print(temperature, 1);
    Serial.print("\t");
    Serial.print(humidity, 1);
    Serial.print("\t");

    uint16_t index = mySGP40.getVocIndex(); //Declare variable that
is to be used to store VOC index.
    Serial.print("vocIndex = ");
    Serial.println(index);

    String dataStr = String(temperature) + " " + String(humidity) + "
" + String(index) + "\n";
    appendFile(SD, "/data.txt", dataStr.c_str());

    delay(10000);
}
```

```
void writeFile(fs::FS &fs, const char *path, const char *message) {  
    Serial.printf("Writing file: %s\n", path);  
  
    File file = fs.open(path, FILE_WRITE);  
    if (!file) {  
        Serial.println("Failed to open file for writing");  
        return;  
    }  
    if (file.print(message)) {  
        Serial.println("File written");  
    } else {  
        Serial.println("Write failed");  
    }  
    file.close();  
}
```

```
void appendFile(fs::FS &fs, const char *path, const char *message) {  
    Serial.printf("Appending to file: %s\n", path);  
  
    File file = fs.open(path, FILE_APPEND);  
    if (!file) {  
        Serial.println("Failed to open file for appending");  
        return;  
    }  
    if (file.print(message)) {  
        Serial.println("Message appended");  
    } else {  
        Serial.println("Append failed");  
    }  
    file.close();  
}
```

Yapılan test kod çalışmaları başarılı olmuştur. Sonraki adımlar

- ESP32 modülünün bir Kablosuz Erişim Noktası (Access Point) oluşturması
- Web sunucu oluşturması
- Sensör verilerini sunması

şeklindedir.

Bir ESP32 mikrodenetleyicisi kullanarak çevresel verileri (sıcaklık, nem ve VOC) ölçer, bu verileri bir SD karta kaydeder ve bir web sunucusu üzerinden bu verilere erişim sağlar. Web sunucusu, ESP32'ye bağlanan cihazlara verileri HTML formatında sunar. Bu sayede, kullanıcılar sensör verilerini gerçek zamanlı olarak görebilirler.

Bu adımlar için test kodu yazmadan sonuç kodu yazılmıştır. Aşağıda sunulmuştur.

```

#include "FS.h"
#include "SD.h"
#include "SPI.h"
#include <DFRobot_SGP40.h>
#include "Wire.h"
#include "SHT31.h"
#define SHT31_ADDRESS 0x44
SHT31 sht;
DFRobot_SGP40 mySGP40; //Declare SGP40.

// Load Wi-Fi library
#include <WiFi.h>

const char *ssid = "ESP32-Access-Point";
const char *password = "123456789";
// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;
// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

bool readSensorEnabled = true;
bool recordDataEnabled = false;
unsigned long capturedTime;
unsigned long previousReadTime = 0;
unsigned long previousRecordTime = 0;

void setup() {
  Serial.begin(115200);
  bool status;
  while (mySGP40.begin(/*duration = */ 10000) != true) {
    Serial.println("Failed to initialize SGP40, please check wiring and cable connections"); //You need to press the
Gravity cable all the way in.
    delay(1000);
  }
  sht.begin();
  sht.read();
  mySGP40.setRHT(/*relativeHumidity = */ sht.getHumidity(), /*temperature = */ sht.getTemperature());
  if (!SD.begin()) {
    Serial.println("Card Mount Failed");
    return;
  }
  uint8_t cardType = SD.cardType();
  if (cardType == CARD_NONE) {
    Serial.println("No SD card attached");
    return;
  }
  writeFile(SD, "/data.txt", "Veriler\n");

  WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();

```



```

    Serial.print("AP IP address: ");
    Serial.println(IP);

    server.begin();
}

float temperature;
float humidity;
uint16_t indexx;

void loop() {
    WiFiClient client = server.available(); // Listen for incoming clients

    if (readSensorEnabled) {
        readSensorEnabled = false;
        sht.read();
        temperature = sht.getTemperature();
        humidity = sht.getHumidity();

        Serial.print(temperature, 1);
        Serial.print("\t");
        Serial.print(humidity, 1);
        Serial.print("\t");

        indexx = mySGP40.getVocIndex(); //Declare variable that is to be used to store VOC index.
        Serial.print("vocIndex = ");
        Serial.println(indexx);

        if (recordDataEnabled) {
            recordDataEnabled = false;
            String dataStr = String(temperature) + " " + String(humidity) + " " + String(indexx) + "\n";
            appendFile(SD, "/data.txt", dataStr.c_str());
        }
    }

    if (client) { // If a new client connects,
        currentTime = millis();
        previousTime = currentTime;
        //Serial.println("New Client."); // print a message out in the serial
        port // make a String to hold incoming data
        from the client

        String currentLine = "";

        while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while the client's connected
            currentTime = millis();
            if (client.available()) { // if there's bytes to read from the client,
                char c = client.read(); // read a byte, then
                Serial.write(c); // print it out the serial monitor
                header += c;
                if (c == '\n') { // if the byte is a newline character
                    // if the current line is blank, you got two newline characters in a row.
                    // that's the end of the client HTTP request, so send a response:
                    if (currentLine.length() == 0) {
                        // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
                        // and a content-type so the client knows what's coming, then a blank line:
                        client.println("HTTP/1.1 200 OK");
                        client.println("Content-type:text/html");
                        client.println("Connection: close");
                    }
                }
            }
        }
    }
}

```



```

client.println();

// Display the HTML web page
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:;\">");
// CSS to style the table
client.println("<style>body { text-align: center; font-family: \"Trebuchet MS\", Arial; }");
client.println("table { border-collapse: collapse; width: 35%; margin-left: auto; margin-right: auto; }");
client.println("tr { padding: 12px; background-color: #0043af; color: white; }");
client.println("tr { border: 1px solid #ddd; padding: 12px; }");
client.println("tr:hover { background-color: #bcbcbc; }");
client.println("td { border: none; padding: 12px; }");
client.println(".sensor { color: black; font-weight: bold; background-color: white; padding: 1px; }");

// Web Page Heading
client.println("</style></head><body><h1>ESP32 with SGP40 and SHT31</h1>");
client.println("<table><tr><th>MEASUREMENT</th><th>VALUE</th></tr>");
client.println("<tr><td>Temp. Celsius</td><td><span class=\"sensor\">");
client.println(temperature);
client.println(" *C</span></td></tr>");
/*client.println("<tr><td>Temp. Fahrenheit</td><td><span class=\"sensor\">");
client.println(1.8 * 25 + 32);
client.println(" *F</span></td></tr>");*/
client.println("<tr><td>Humidity</td><td><span class=\"sensor\">");
client.println(humidity);
client.println("%</span></td></tr>");
/*client.println("<tr><td>Approx. Altitude</td><td><span class=\"sensor\">");
client.println(2000);
client.println(" m</span></td></tr>");*/
client.println("<tr><td>Air Quality</td><td><span class=\"sensor\">");
client.println(index);
client.println("</span></td></tr>");
client.println("</body></html>");

// The HTTP response ends with another blank line
client.println();
// Break out of the while loop
break;
} else { // if you got a newline, then clear currentLine
currentLine = "";
}
} else if (c != '\r') { // if you got anything else but a carriage return character,
currentLine += c; // add it to the end of the currentLine
}
}
}

// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}

capturedTime = millis();

```

```

    if (capturedTime - previousReadTime >= 1000) {
        previousReadTime = capturedTime;
        readSensorEnabled = true;
    }

    if (capturedTime - previousRecordTime >= 10000) {
        previousRecordTime = capturedTime;
        recordDataEnabled = true;
    }
}

void writeFile(fs::FS &fs, const char *path, const char *message) {
    Serial.printf("Writing file: %s\n", path);

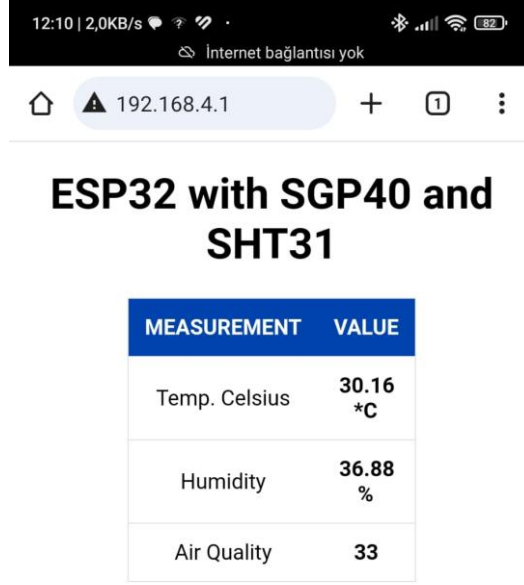
    File file = fs.open(path, FILE_WRITE);
    if (!file) {
        Serial.println("Failed to open file for writing");
        return;
    }
    if (file.print(message)) {
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
    file.close();
}

void appendFile(fs::FS &fs, const char *path, const char *message) {
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if (!file) {
        Serial.println("Failed to open file for appending");
        return;
    }
    if (file.print(message)) {
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}

```

Mobil cihazdan alınan ekran görüntüsü aşağıdaki gibidir.



MEASUREMENT	VALUE
Temp. Celsius	30.16 °C
Humidity	36.88 %
Air Quality	33

3. Sonuç

4. Çıktılar

Bu proje kapsamında Mobil Hava Kalitesi izleme sistemi başarılı bir şekilde tasarlanmış ve gerçekleştirilmiştir. Proje çıktısı yürütücünün Bitirme Tezi (Mezun olması için gerekli Proje çalışması) olmuştur.

5. Proje ile ilgili harcama kalemleri hakkında ayrıntılı bilgi

No	Harcama Kalemi	Gerekçe
1	ESP32	Sensör verilerini okumak ve Kablosuz sunmak için kullanılmıştır.
2	Lehim Teli Isıyla daralan makaron Muhtelif renkte kablo	Elektronik kartların birleştirilmesi için gerekli olan malzemelerdir.
3	SD kart	Verileri depolamak için gereklidir..
4	SGP40 Hava Kalitesi Sensörü ve SHT31 Sıcaklık ve Nem Sensörü Modülü	Sistemin ana bileşenlerini oluşturur. Hava kalitesini sıcaklığı ve nemi ölçmek için gerekli sensörlerdir.
5	TP4056 Korumalı LİTYUM BATARYA Şarj Modülü	Mobil sistemin bataryasını şarj etmek ve gerekli gerilim düzeyini sağlamak için kullanılmıştır.
6	18650 Tekli Pil Yuvası	18650 tipinde Li-İon pili devrelere bağlamak üzere gerekli olan yuvadır.
7	Raspberry Pi Zero 2 W	ESP32 performansı proje önerilirken kestirilemediğinden, işlem

		gücü ESP32'den daha yüksek olan bu modül satın alınarak denenmiştir.
8	Li-ion Şarjlı Pil	Mobil sistemin enerji kaynağıdır.
9	Raspberry Pi 5	ESP32 performansı proje önerilirken kestirilemediğinden, işlem gücü ESP32'den daha yüksek olan bu modül satın alınarak denenmiştir.
10	OLED Grafik Lcd Ekran	Web sunucun başarısız olma ihtimaline karşı B planı olarak alınmış bir önlemdir. Cihazdaki sensörlerin durumunu gözlemlemek için alınmıştır.

PROJE YÜRÜTÜCÜSÜNÜN

ADI – SOYADI - İMZA

DANIŞMANIN

ADI – SOYADI - İMZA

Erhan ÖZDOĞAN	Dr. Öğr. Üyesi Mehmet Kürşat YALÇIN

Tarih :