ДОДАТОК А

Код програми для ESP32

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Adafruit_AHTX0.h>
#include "ScioSense_ENS160.h"
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESP32Servo.h>
#include <ArduinoJson.h>

const char* ssid = "_ShurikZz";
const char* password = "ShurikZz";

LiquidCrystal_I2C lcd(0x27, 16, 2);
static const int HMout = 13;
static const int HMin = 12;
static const int Flamesens = 4;
static const int Smokesens = 5;
static const int MotoPlus = 14;
static const int MotoMinus = 27;
static const int LouversPin = 18;
static const int CondiPin = 17;

Servo louvers;
Servo condi;
Adafruit_AHTX0 aht;
ScioSense_ENS160 ens160(ENS160_I2CADDR_1);
WebServer server(80);

int flame = HIGH;
int smoke;
int tempC;
int humidity;
int AQI;
int TVOC = 55;
int eCO2;
int lcdCount = 1;
int impulseLogic = 1;
int louversLogic = 1;
int tempCThreshold = 22;
int humidityThreshold = 55;
bool onOffBool = false;

void setup() {
  Serial.begin(115200);
  louvers.attach(LouversPin);
  condi.attach(CondiPin);
  lcd.init();
  lcd.backlight();
  lcd.clear();
```

```
  pinMode(HMin, INPUT);
  pinMode(HMout, OUTPUT);
  pinMode(Flamesens, INPUT);
  pinMode(Smokesens, INPUT);
  pinMode(MotoPlus, OUTPUT);
  pinMode(MotoMinus, OUTPUT);

  ENS160plusAHT21Connect();

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  server.on("/getSensorData", HTTP_GET, handleGetSensorData);
  server.on("/setTempCThreshold", HTTP_POST, SetTempCThresholdFromApplication);
  server.on("/setHumidityThreshold", HTTP_POST, SetHumidityThresholdFromApplication);
  server.on("/setOnOffBool", HTTP_POST, SetOnOffBoolFromApplication);

  server.begin();
}

void loop() {
  server.handleClient();
  elementControl();
}

void handleGetSensorData() {
  isFlame();
  airQualitySensor();
  lcd.clear();
  LCDisplay();
  StaticJsonDocument<200> jsonDoc;
  jsonDoc["Flame"] = flame;
  jsonDoc["Smoke"] = smoke;
  jsonDoc["TemperatureC"] = tempC;
  jsonDoc["HumiditypH"] = humidity;
  jsonDoc["AQI"] = AQI;
  jsonDoc["TVOC"] = TVOC;
  jsonDoc["eCO2"] = eCO2;

  String jsonResponse;
  serializeJson(jsonDoc, jsonResponse);

  server.send(200, "application/json", jsonResponse);

}

void SetTempCThresholdFromApplication () {
   if (!server.hasArg("tempCThreshold")) {
    server.send(400, "application/json", "{\"status\":\"error\",\"message\":\"Missing parameters\"}");
    return;
```

```
  }
  tempCThreshold = server.arg("tempCThreshold").toInt();
  Serial.print("tempCThreshold: ");
  Serial.println(tempCThreshold);
  server.send(200, "application/json", "{\"status\":\"ok\"}");
}
void SetHumidityThresholdFromApplication () {
   if (!server.hasArg("humidityThreshold")) {
    server.send(400, "application/json", "{\"status\":\"error\",\"message\":\"Missing parameters\"}");
    return;
  }
  humidityThreshold = server.arg("humidityThreshold").toInt();
  Serial.print("humidityThreshold: ");
  Serial.println(humidityThreshold);
  server.send(200, "application/json", "{\"status\":\"ok\"}");
}
void SetOnOffBoolFromApplication () {
   if (!server.hasArg("onOffBool")) {
    server.send(400, "application/json", "{\"status\":\"error\",\"message\":\"Missing parameters\"}");
    return;
  }
  onOffBool = server.arg("onOffBool") == "1";
  Serial.print("onOffBool: ");
  Serial.println(onOffBool);
  server.send(200, "application/json", "{\"status\":\"ok\"}");
}

void ENS160plusAHT21Connect() {
  Serial.println("ENS160 - Digital air quality sensor");
  Serial.print("ENS160...");
  ens160.begin();
  Serial.println(ens160.available() ? "done." : "failed!");
  if (ens160.available()) {
    Serial.print("\tRev: "); Serial.print(ens160.getMajorRev());
    Serial.print("."); Serial.print(ens160.getMinorRev());
    Serial.print("."); Serial.println(ens160.getBuild());
    Serial.print("\tStandard mode ");
    Serial.println(ens160.setMode(ENS160_OPMODE_STD) ? "done." : "failed!");

    Serial.println("Adafruit AHT10/AHT20");
    if (!aht.begin()) {
      Serial.println("Could not find AHT? Check wiring");
      while (1) delay(10);
    }
    Serial.println("AHT10 or AHT20 found");
  }
}

void isFlame() {
  flame = digitalRead(Flamesens);
  smoke = digitalRead(Smokesens);
  // Serial.print("Flame: ");
  // Serial.print(flame);
  // Serial.print(" Smoke: ");
  // Serial.print(smoke);Serial.print(", ");
  if (flame == LOW) {
```

```
    lcdCount = 4;
  }
}

void airQualitySensor() {
  sensors_event_t humidity1, temp;
  aht.getEvent(&humidity1, &temp);
  tempC = (temp.temperature) - 4;
  humidity = (humidity1.relative_humidity);
  // Serial.print("Temperature: ");
  // Serial.print(tempC);
  // Serial.print(" C, ");
  // Serial.print("Humidity: ");
  // Serial.print(humidity);
  // Serial.print("%, ");


  AQI = ens160.getAQI();
  TVOC = ens160.getTVOC();
  eCO2 = ens160.geteCO2();

  ens160.set_envdata(tempC, humidity);
  ens160.measure(true);
  ens160.measureRaw(true);

  Serial.print("AQI: "); Serial.print(AQI); Serial.print(", ");
  Serial.print("TVOC: "); Serial.print(TVOC); Serial.print(" ppb, ");
  Serial.print("eCO2: "); Serial.print(eCO2); Serial.println(" ppm");

}

void LCDisplay() {
  if (smoke == LOW) {
    lcd.setCursor(3, 0);
    lcd.print("Attention!");
    lcd.setCursor(0, 1);
    lcd.print("Fire+smoke alarm");
  }
  else if (flame == LOW) {
    lcd.setCursor(3, 0);
    lcd.print("Attention!!!");
    lcd.setCursor(2, 1);
    lcd.print("Fire alarm!!!");
  }
  else{
    lcd.print(tempC);
    lcd.print("C ");
    lcd.print(humidity);
    lcd.print("%");
    lcd.print(" AQI:");
    lcd.print(AQI);
    lcd.setCursor(0, 1);
    lcd.print(TVOC);
    lcd.print("ppb ");
    lcd.print(eCO2);
    lcd.print("ppm");
```

```
  }
}

void elementControl () {
  if (onOffBool && flame == 1)
  {
    if (AQI > 2) {
      fanOn();
      louversUp();
    } else {
      fanOff();
    }
    if (TVOC > 50)
    {
      louversUp();
    }
    else {
      louversDown();
    }
    if (tempC > tempCThreshold) {
      conditionOn();
    } else if (tempC < tempCThreshold) {
      warmOn();
    } else {
      condiOff();
    }
    if (humidity < humidityThreshold) {
      HumidifierOn();
    } else {
      HumidifierOff();
    }
  }
  else if (flame == 0){
    fanOff();
    HumidifierOff();
    condiOff();
    louversDown();
  }
  else
  {
    fanOff();
    HumidifierOff();
    condiOff();
    louversUp();
  }
}
void fanOn() {
  digitalWrite(MotoPlus, HIGH);
  digitalWrite(MotoMinus, LOW);
}

void fanOff() {
  digitalWrite(MotoPlus, LOW);
  digitalWrite(MotoMinus, LOW);
}
void conditionOn() {
```

```
  condi.write(120);
}
void warmOn() {
  condi.write(85);
}
void condiOff() {
  condi.write(90);
}
void louversUp() {
  if (louversLogic == 1) {
    louvers.write(180);
    delay(400);
    louvers.write(90);
    louversLogic = 0;
  }
  else {louvers.write(90);}
}
void louversDown() {
  if (louversLogic == 0) {
    louvers.write(0);
    delay(400);
    louvers.write(90);
    louversLogic = 1;
  }
  else {louvers.write(90);}
}
void HumidifierOn() {
  if (impulseLogic == 1) {
    digitalWrite(HMout, HMin);
    delay(400);
    digitalWrite(HMout, LOW);
    impulseLogic = 0;
  } else
  {
    digitalWrite(HMout, LOW);
  }
}
void HumidifierOff() {
  if (impulseLogic == 0) {
    digitalWrite(HMout, HMin);
    delay(200);
    digitalWrite(HMout, LOW);
    delay(200);
    digitalWrite(HMout, HMin);
    delay(200);
    digitalWrite(HMout, LOW);
    impulseLogic = 1;
  } else
  {
    digitalWrite(HMout, LOW);
  }
}
```

ДОДАТОК Б

Код програми управління системою

Лістинг Б.1 – файл MainWindow.axaml

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="using:Air_quality.ViewModels"
                xmlns:local="clr-namespace:Air_quality.Classes;assembly=Air quality"
                xmlns:gif="clr-namespace:AvaloniaGif;assembly=AvaloniaGif"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
                xmlns:views="clr-namespace:Air_quality.Views"
        mc:Ignorable="d" d:DesignWidth="1450" d:DesignHeight="950"
                MinWidth="1450" MinHeight="950"
        x:Class="Air_quality.Views.MainWindow"
        x:DataType="vm:MainWindowViewModel"
        Icon="/Assets/wind-ico.png"
        Title="Air quality"
                TransparencyLevelHint="AcrylicBlur"
        Background="Transparent"
                ExtendClientAreaToDecorationsHint="True"
        ExtendClientAreaChromeHints="Default">

    <Design.DataContext>
        <vm:MainWindowViewModel/>
    </Design.DataContext>

            <Window.Styles>
                <StyleInclude Source="/Styles/Styles.axaml"></StyleInclude>
            </Window.Styles>

            <Window.Resources>
                <local:AQIToColorConverter x:Key="AQIToColorConverter"/>
                <local:AQIToTextConverter x:Key="AQIToTextConverter"/>
                <local:PercentageToWidthConverter x:Key="PercentageToWidthConverter"/>
                <local:GreaterThanToVisibilityConverterFan
x:Key="GreaterThanToVisibilityConverterFan"/>
                <local:GreaterThanToVisibilityConverterCondition
x:Key="GreaterThanToVisibilityConverterCondition"/>
                <local:GreaterThanToVisibilityConverterLouvers
x:Key="GreaterThanToVisibilityConverterLouvers"/>
                <local:LessThanToVisibilityConverterFan
x:Key="LessThanToVisibilityConverterFan"/>
                <local:LessThanToVisibilityConverterWarm
x:Key="LessThanToVisibilityConverterWarm"/>
                <local:LessThanToVisibilityConverterHumidity
x:Key="LessThanToVisibilityConverterHumidity"/>
                <local:FireAlarmLogic x:Key="FireAlarmLogic"/>
            </Window.Resources>
```

```
<!--Main Page-->
<Grid RowDefinitions="Auto, *, *, Auto">
  <!--Menu-->
  <Grid Grid.Row="0" Margin="10 1" ColumnDefinitions="Auto, *">
    <TextBlock LetterSpacing="3" IsHitTestVisible="False" Grid.ColumnSpan="2" FontSize="24" HorizontalAlignment="Center" VerticalAlignment="Center">Automation of air quality</TextBlock>

  <Button Width="90" Margin="15" Classes="ON" x:Name="ON" Command="{Binding OnOffCommand}" Click="ButtonON_Click" Background="DarkGreen" >ON</Button>
  <Button Width="90" Margin="15" Classes="OFF" x:Name="OFF" Command="{Binding OnOffCommand}" IsVisible="False" Click="ButtonOFF_Click" Background="DarkRed" >OFF</Button>
  </Grid>
  <!--ON/OFF|Avtivity-->
  <Grid Grid.Row="1" Margin="10 10 10 0">
    <TextBlock ZIndex="1" Opacity="0.8" FontSize="150" VerticalAlignment="Center" HorizontalAlignment="Center" Foreground="Red" Text="FIRE ALARM!!!">
      <TextBlock.IsVisible>
        <MultiBinding Converter="{StaticResource FireAlarmLogic}">
          <Binding Path="CurrentSensorData.Flame" />
        </MultiBinding>
      </TextBlock.IsVisible>
    </TextBlock>
    <!--UI-->
    <Border Margin="10" CornerRadius="10" ClipToBounds="True" Opacity="150" Background="Gray">
      <Grid>
        <Grid ColumnDefinitions="*, *, *, *, *">
          <gif:GifImage StretchDirection="Both" ToolTip.Tip="Stop" Stretch="Uniform" Opacity="7" SourceUri="avares://Air quality/Assets/Fan.gif" >
            <gif:GifImage.IsVisible>
              <MultiBinding Converter="{StaticResource LessThanToVisibilityConverterFan}">
                <Binding Path="CurrentSensorData.AQI" />
                <Binding Path="AQIThreshold" />
                <Binding Path="OnOffBool" />
              </MultiBinding >
            </gif:GifImage.IsVisible>
          </gif:GifImage>
          <gif:GifImage StretchDirection="Both" ToolTip.Tip="Working..." AutoStart="True" Stretch="Uniform" Opacity="7" SourceUri="avares://Air quality/Assets/Fan.gif" >
            <gif:GifImage.IsVisible>
              <MultiBinding Converter="{StaticResource GreaterThanToVisibilityConverterFan}">
                <Binding Path="CurrentSensorData.AQI" />
                <Binding Path="AQIThreshold" />
```

```
                                        <Binding Path="OnOffBool" />
                                    </MultiBinding >
                                </gif:GifImage.IsVisible>
                            </gif:GifImage>
                            <TextBlock Text="Fan" Margin="7" />

                            <Image Grid.Column="1" ToolTip.Tip="Stop"
StretchDirection="Both" Stretch="Uniform" Opacity="7" Source="avares://Air
quality/Assets/ConditionStatic.png" ></Image>
                            <gif:GifImage Grid.Column="1" ToolTip.Tip="Working..."
AutoStart="True" StretchDirection="Both" Stretch="Uniform" Opacity="7"
SourceUri="avares://Air quality/Assets/Condition.gif" >
                                <gif:GifImage.IsVisible>
                                    <MultiBinding Converter="{StaticResource
GreaterThanToVisibilityConverterCondition}">
                                        <Binding
Path="CurrentSensorData.TemperatureC"  />
                                        <Binding
Path="TemperatureThreshold" />
                                        <Binding Path="OnOffBool" />
                                    </MultiBinding >
                                </gif:GifImage.IsVisible>
                            </gif:GifImage>
                        <TextBlock Grid.Column="1" Text="Condition" Margin="7" />
                            <Image Grid.Column="2" ToolTip.Tip="Stop" Opacity="7"
Source="avares://Air quality/Assets/WarmStatic.png" />
                            <gif:GifImage Grid.Column="2" ToolTip.Tip="Working..."
StretchDirection="Both" Stretch="Uniform" Opacity="7" AutoStart="True"
SourceUri="avares://Air quality/Assets/Warm.gif" >
                                <gif:GifImage.IsVisible>
                                    <MultiBinding Converter="{StaticResource
LessThanToVisibilityConverterWarm}">
                                        <Binding
Path="CurrentSensorData.TemperatureC"  />
                                        <Binding
Path="TemperatureThreshold" />
                                        <Binding Path="OnOffBool" />
                                    </MultiBinding >
                                </gif:GifImage.IsVisible>
                            </gif:GifImage>
                            <TextBlock Grid.Column="2" Text="Heat" Margin="7" />


                            <Image Grid.Column="3" Margin="0 15 0 0"
ToolTip.Tip="Stop" Source="{SvgImage /Assets/Ventilation_louvres_closed.svg}" />
                            <Image Grid.Column="3" ToolTip.Tip="Working..."
Margin="0 15 0 0" IsVisible="True" Source="{SvgImage /Assets/Ventilation_louvres_open.svg}"
>
                                <Image.IsVisible>
                                    <MultiBinding Converter="{StaticResource
GreaterThanToVisibilityConverterLouvers}">
```

```
                                                    <Binding
Path="CurrentSensorData.AQI"  />
                                                    <Binding Path="AQIThreshold" />
                                                    <Binding Path="OnOffBool" />
                                                </MultiBinding >
                                            </Image.IsVisible>
                                        </Image>
                                        <TextBlock  Grid.Column="3"  Text="Ventilation  louvres"
Margin="7" />



                                        <TextBlock        Grid.Column="4"        Text="Humidity"
Margin="7" />
                                        <PathIcon Width="400" ToolTip.Tip="Stop" Height="150"
Data="{StaticResource humidity}" Grid.Column="4" />
                                        <PathIcon       Width="400"       ToolTip.Tip="Working..."
Height="150" Foreground="SteelBlue" Data="{StaticResource humidity}" Grid.Column="4" >
                                            <PathIcon.IsVisible>
                                                <MultiBinding  Converter="{StaticResource
LessThanToVisibilityConverterHumidity}">
                                                    <Binding
Path="CurrentSensorData.HumiditypH" />
                                                    <Binding  Path="HumidityThreshold"
/>
                                                    <Binding Path="OnOffBool" />
                                                </MultiBinding >
                                            </PathIcon.IsVisible>
                                        </PathIcon>
                                    </Grid>
                                </Grid>
                            </Border>
                        </Grid>
                        <!--Parameters|Info-->
                        <Grid Grid.Row="2" Margin="10 0 10 10"  ColumnDefinitions="*, *">
                            <!--Left-->
                            <Border Margin="10" CornerRadius="10" ClipToBounds="True" Opacity="150"
Background="Gray">
                                <Grid RowDefinitions="0.5*, *, *, *, *" ColumnDefinitions="AUTO, 2*,
*, AUTO, *, AUTO" >
                                    <TextBlock Text="Indoors" Grid.ColumnSpan="2" FontSize="20"
Margin="10 10 5 0"/>
                                    <PathIcon      Grid.Column="0"      Grid.Row="1"      Width="100"
Height="50" Data="{StaticResource temperature}" />
                                    <TextBlock  Grid.Column="1"  ToolTip.Tip="This  is  the  air
temperature"          Grid.Row="1"          FontSize="35"          HorizontalAlignment="Left"
VerticalAlignment="Center">Temperature:</TextBlock>
                                    <TextBlock          Text="Current"          FontSize="25"
HorizontalAlignment="Left"  VerticalAlignment="Center"  Grid.Row="0"  Grid.Column="2"
></TextBlock>
```

```
                    <TextBlock Text="Desired" FontSize="25" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Row="0" Grid.Column="4" ></TextBlock>
                    <TextBlock ToolTip.Tip="{Binding CurrentSensorData.TemperatureC, StringFormat='Temperature is {0} °C', FallbackValue='Temperature is 22 °C'}" Text="{Binding CurrentSensorData.TemperatureC, StringFormat=' {0}°C', FallbackValue=' 22°C'}" FontSize="35" HorizontalAlignment="Left" VerticalAlignment="Center" Grid.Row="1" Grid.Column="2" ></TextBlock>

                    <Button Command="{Binding DecreaseTemperatureThresholdCommand}" FontSize="25" Opacity="144" Grid.Column="3" Grid.ColumnSpan="2" Grid.Row="1" Content="-" />
                    <TextBlock ToolTip.Tip="{Binding TemperatureThreshold, StringFormat='Desired temperature is {0} °C', FallbackValue='Desired temperature'}" Text="{Binding TemperatureThreshold}" FontSize="45" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="4" Grid.Row="1"></TextBlock>
                    <Button Command="{Binding IncreaseTemperatureThresholdCommand}" HorizontalAlignment="Right" Margin="0 0 5 0" FontSize="25" Opacity="144" Grid.Column="4" Grid.ColumnSpan="2" Grid.Row="1" Content="+" />
                    <PathIcon Width="100" Height="50" Data="{StaticResource humidity}" Grid.Row="2" />
                    <TextBlock FontSize="35" ToolTip.Tip="This is the air humidity" HorizontalAlignment="Left" VerticalAlignment="Center" Grid.Row="2" Grid.Column="1">Humidity:</TextBlock>
                    <TextBlock Text="{Binding CurrentSensorData.HumiditypH, StringFormat=' {0}%', FallbackValue=' 70%'}" FontSize="35" HorizontalAlignment="Left" VerticalAlignment="Center" Grid.Row="2" Grid.Column="2" ></TextBlock>
                    <Button Command="{Binding DecreaseHumidityThresholdCommand}" FontSize="25" Opacity="144" Grid.Column="3" Grid.ColumnSpan="2" Grid.Row="2" Content="-" />
                    <TextBlock ToolTip.Tip="{Binding HumidityThreshold, StringFormat='Desired humidity is {0}%'}" Text="{Binding HumidityThreshold}" FontSize="45" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Row="2" Grid.Column="4"></TextBlock>
                    <Button Command="{Binding IncreaseHumidityThresholdCommand}" HorizontalAlignment="Right" Margin="0 0 5 0" FontSize="25" Opacity="144" Grid.Column="4" Grid.ColumnSpan="2" Grid.Row="2" Content="+" />
                    <Grid RowDefinitions="*, *" ColumnDefinitions=" *, *" Grid.RowSpan="2" Grid.Row="3" Grid.ColumnSpan="6" Margin="15" >
                    <TextBlock Grid.Column="0" ToolTip.Tip="Air Quality Index
                                | 5 Dangerous
                                | 4 Bad
                                | 3 Normal
                                | 2 Good
                                | 1 Excellent "
                                Text="{Binding CurrentSensorData.AQI, StringFormat='AQI: {0}', FallbackValue='AQI: 0'}" FontSize="35" HorizontalAlignment="Center" VerticalAlignment="Center"/>
```

```
                        <TextBlock Grid.Column="1" ToolTip.Tip="Convert air quality index into words" Text="{Binding CurrentSensorData.AQI, Converter={StaticResource AQIToTextConverter}, FallbackValue=' unknown '}" Background="{Binding CurrentSensorData.AQI, Converter={StaticResource AQIToColorConverter}, FallbackValue='Gray'}" FontSize="35" HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        <Rectangle Grid.RowSpan="2" Grid.ColumnSpan="2" Margin="40 0"  Height="5" Fill="White" />
<TextBlock Grid.Column="0" Grid.Row="2" ToolTip.Tip="Total Volatile Organic Compounds
                        | More than 6000 headaches and nerve problems
                                | 750-6000 feeling of anxiety and headaches
                                | 50-750 Feeling of anxiety and discomfort
                                | Less than 50 has no effect       "
="{Binding CurrentSensorData.TVOC, StringFormat='TVOC: {0} ppb', FallbackValue='TVOC: 800 ppb'}" FontSize="35" HorizontalAlignment="Center" VerticalAlignment="Center"/>
<TextBlock Grid.Column="1" Grid.Row="2" ToolTip.Tip="Equivalent of carbon dioxide
                        | 21500 Dangerous
                                | 1000-1500 Bad
                                | 800-1000 Normal
                                | 600-800 Good
                                | 400-600 Excellent "
                                                Text="{Binding CurrentSensorData.eCO2, StringFormat='eCO2: {0} ppm', FallbackValue='eCO2: 1500 ppm'}" FontSize="35" HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        </Grid>
                    </Grid>
                </Border>
                <!--Right-->
                <Border Grid.Column="1" Margin="10" CornerRadius="10" ClipToBounds="True" Opacity="150" Background="Gray">
                    <Grid RowDefinitions="4*, AUTO, AUTO, AUTO, AUTO" ColumnDefinitions="*, *">
                        <Grid Grid.ColumnSpan="2" RowDefinitions="*,*,*,*" ColumnDefinitions="*,*" >
                            <Border BorderThickness="2" Margin="0 0 0 2" BorderBrush="White" CornerRadius="10" Grid.ColumnSpan="2" Grid.RowSpan="4" />
                            <TextBlock Text="Weather outside" Grid.ColumnSpan="2" FontSize="20" Margin="10 10 5 0"/>
                            <TextBlock Grid.Row="0" Grid.RowSpan="2" VerticalAlignment="Center" HorizontalAlignment="Center" ToolTip.Tip="City" Text="{Binding CityName, StringFormat=' {0}'}" FontSize="35" />
                            <TextBlock Grid.Row="0" Grid.RowSpan="2" Grid.Column="1" ToolTip.Tip="{Binding WeatherData.Main.Temp, StringFormat='It\'s {0} degrees outside'}" FontSize="35" Text="{Binding WeatherData.Main.Temp, StringFormat=' {0}°C', FallbackValue='None'}" VerticalAlignment="Center" HorizontalAlignment="Center" Margin="2"/>
<TextBlock Grid.Row="3" FontSize="25" Text="{Binding WeatherData.Weather[0].Description, FallbackValue='None'}" HorizontalAlignment="Center" Margin="2"/>
<TextBlock Grid.Row="2" Grid.Column="0" FontSize="25" Text="{Binding Humidity, StringFormat='Humidity: {0}%'}" HorizontalAlignment="Center" Margin="2"/>
<TextBlock Grid.Row="2" Grid.Column="2" FontSize="25" Text="{Binding Pressure, StringFormat='Pressure: {0} hPa'}" HorizontalAlignment="Center" Margin="2"/>
```

```
<TextBlock Grid.Row="3" Grid.Column="3" FontSize="25" Text="{Binding WindSpeed, StringFormat='Wind Speed: {0} m/s'}" HorizontalAlignment="Center" Margin="2"/>
                                </Grid>
                                <!--Bags-->
<TextBlock Grid.Row="1" FontSize="20" Margin="15 5" >Dust collector fullness</TextBlock>
                                <Button Classes="ButtonReset" Grid.Row="1" Grid.Column="1" Command="{Binding ResetTimer1Command}" >Reset</Button>
                                <Grid Grid.Row="2" Grid.ColumnSpan="2" Height="40" >
                                        <ProgressBar Grid.Column="2" Minimum="0" Maximum="1" Value="{Binding FillPercentage1}" Height="20" Margin="15" />
                    <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center" Foreground="Black" Text="{Binding ElapsedPercentage1, StringFormat=' {0:F1}%'}" />
                                </Grid>
                                <!--Filter-->
<TextBlock Grid.Row="3" FontSize="20" Margin="15 5" >Filter contamination</TextBlock>
                                <Button Classes="ButtonReset" Grid.Row="3" Grid.Column="1" Command="{Binding ResetTimer2Command}" >Reset</Button>
                                <Grid Grid.Row="4" Grid.ColumnSpan="2" Height="40" >
                                        <ProgressBar Grid.Column="2" Minimum="0" Maximum="1" Value="{Binding FillPercentage2}" Height="20" Margin="15" />
                                        <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center" Foreground="Black" Text="{Binding ElapsedPercentage2, StringFormat=' {0:F1}%'}" x:Name="FiltersContamination" />
                                </Grid>
                        </Grid>
                </Border>
            </Grid>
        </Grid>
</Window>
```

Лістинг Б.2 – файл MainWindow.axaml.cs

```
using Air_quality.ViewModels;
using Avalonia.Controls;
using System;
using System.ComponentModel;
using System.Timers;
namespace Air_quality.Views
{
   public partial class MainWindow : Window
   {
      private MainWindowViewModel ViewModel;
      private static Timer timer1;

      public MainWindow()
      {
         InitializeComponent();
         ViewModel = new MainWindowViewModel();
         DataContext = ViewModel;
         this.Closing += OnClosing;

         timer1 = new Timer(3000);
         timer1.Elapsed += IsFlame;
```

```
            timer1.Start();
        }
        private void IsFlame(object sender, ElapsedEventArgs e)
        {
            if (ViewModel.isFlame == 0)
            {
                OFF.IsVisible = false;
                ON.IsVisible = true;
                ViewModel.StopTimer1();
                ViewModel.StopTimer2();
            }
        }
        private void ButtonON_Click(object? sender, Avalonia.Interactivity.RoutedEventArgs e)
        {
            ON.IsVisible = false;
            OFF.IsVisible = true;
            ViewModel.StartTimer1();
            ViewModel.StartTimer2();
        }
        private void ButtonOFF_Click(object? sender, Avalonia.Interactivity.RoutedEventArgs e)
        {
            OFF.IsVisible = false;
            ON.IsVisible = true;
            ViewModel.StopTimer1();
            ViewModel.StopTimer2();
        }
        private void OnClosing(object sender, CancelEventArgs e)
        {
            if (DataContext is MainWindowViewModel viewModel)
            {
                viewModel.SaveSettings();
            }
        }
    }
}
```

Лістинг Б.3 – файл MainWindowViewModel.cs

```
using System;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using System.Timers;
using System.Windows.Input;
using ReactiveUI;
using Air_quality.Classes;
using System.IO.IsolatedStorage;
using System.IO;
using System.Collections.Generic;
namespace Air_quality.ViewModels
{
    public class MainWindowViewModel : ReactiveObject
    {
```

```
public event EventHandler ElapsedPercentage1ThresholdReached;
public event EventHandler ElapsedPercentage2ThresholdReached;
private static Timer timer1;
private static Timer timer2;
private Timer updateTimer;
private static int totalSeconds1 = 55400;
private static int totalSeconds2 = 86400;
private int elapsedSeconds1;
private int elapsedSeconds2;
private SensorData currentSensorData;
private bool onOffBool = false;
private bool preOnOffBool = false;
private int aQIThreshold = 2;
private int temperatureThreshold;
private int humidityThreshold;
public int isFlame = 1;
public int ElapsedSeconds1
{
   get => elapsedSeconds1;
   set
   {
      this.RaiseAndSetIfChanged(ref elapsedSeconds1, value);
      this.RaisePropertyChanged(nameof(FillPercentage1));
      this.RaisePropertyChanged(nameof(ElapsedPercentage1));
   }
}
public int ElapsedSeconds2
{
   get => elapsedSeconds2;
   set
   {
      this.RaiseAndSetIfChanged(ref elapsedSeconds2, value);
      this.RaisePropertyChanged(nameof(FillPercentage2));
      this.RaisePropertyChanged(nameof(ElapsedPercentage2));
   }
}
public double FillPercentage1 => (double)ElapsedSeconds1 / totalSeconds1;
public double FillPercentage2 => (double)ElapsedSeconds2 / totalSeconds2;
public double ElapsedPercentage1 => (double)ElapsedSeconds1 / totalSeconds1 * 100;
public double ElapsedPercentage2 => (double)ElapsedSeconds2 / totalSeconds2 * 100;
public SensorData CurrentSensorData
{
   get => currentSensorData;
   set => this.RaiseAndSetIfChanged(ref currentSensorData, value);
}
private WeatherResponse weatherData;
public WeatherResponse WeatherData
{
   get => weatherData;
   set => this.RaiseAndSetIfChanged(ref weatherData, value);
}
private string cityName;
```

```
public string CityName
{
    get => cityName;
    set
    {
        this.RaiseAndSetIfChanged(ref cityName, value);
        if (!string.IsNullOrEmpty(cityName))
        {
            _ = GetWeatherDataAsync(cityName);
        }
    }
}
private int humidity;
public int Humidity
{
    get => humidity;
    set => this.RaiseAndSetIfChanged(ref humidity, value);
}
private double pressure;
public double Pressure
{
    get => pressure;
    set => this.RaiseAndSetIfChanged(ref pressure, value);
}

private double windSpeed;
public double WindSpeed
{
    get => windSpeed;
    set => this.RaiseAndSetIfChanged(ref windSpeed, value);
}
public bool OnOffBool
{
    get => onOffBool;
    set
    {
        this.RaiseAndSetIfChanged(ref onOffBool, value);
        _ = SendOnOffBoolToESP();
    }
}
public int AQIThreshold
{
    get => aQIThreshold;
}
public int TemperatureThreshold
{
    get => temperatureThreshold;
    set
    {
        this.RaiseAndSetIfChanged(ref temperatureThreshold, value);
        _ = SendTemperatureThresholdToESP();
    }
```

```
        }
        public int HumidityThreshold
        {
            get => humidityThreshold;
            set
            {
                this.RaiseAndSetIfChanged(ref humidityThreshold, value);
                _ = SendHumidityThresholdToESP();
            }
        }
        private static readonly string apiKey = "12f91a8e85221c8330c764b9b9073817";
        private static readonly string baseUrl = "http://api.openweathermap.org/data/2.5/weather";
        private static readonly string ESPBaseUrl = "http://192.168.182.41"; // Замените на IP адрес
вашего ESP
        public ICommand ConnectCommand { get; }
        public ICommand ResetTimer1Command { get; }
        public ICommand ResetTimer2Command { get; }
        public ICommand OnOffCommand { get; }
        public ICommand IncreaseTemperatureThresholdCommand { get; }
        public ICommand DecreaseTemperatureThresholdCommand { get; }
        public ICommand IncreaseHumidityThresholdCommand { get; }
        public ICommand DecreaseHumidityThresholdCommand { get; }
        public MainWindowViewModel()
        {
            // Ініціалізація компонентів, таймерів, команд тощо
            timer1 = new Timer(1000);
            timer1.Elapsed += OnTimedEvent1;
            timer1.AutoReset = true;
            timer2 = new Timer(1000);
            timer2.Elapsed += OnTimedEvent2;
            timer2.AutoReset = true;
            ResetTimer1Command = ReactiveCommand.Create(ResetTimer1);
            ResetTimer2Command = ReactiveCommand.Create(ResetTimer2);
            OnOffCommand = ReactiveCommand.Create(OnOffTrue);
            IncreaseTemperatureThresholdCommand = ReactiveCommand.Create(() =>
TemperatureThreshold += 1);
            DecreaseTemperatureThresholdCommand = ReactiveCommand.Create(() =>
TemperatureThreshold -= 1);
            IncreaseHumidityThresholdCommand = ReactiveCommand.Create(() =>
HumidityThreshold += 1);
            DecreaseHumidityThresholdCommand = ReactiveCommand.Create(() =>
HumidityThreshold -= 1);
            CityName = "Tartu";
            LoadSettings();
            updateTimer = new Timer(3000);
            updateTimer.Elapsed += async (sender, e) => await UpdateSensorDataAsync();
            updateTimer.Elapsed += IsFlame;
            updateTimer.Start();
            SendOnOffBoolToESP();
        }
        private void OnTimedEvent1(object sender, ElapsedEventArgs e)
        {
```

```
    if (ElapsedSeconds1 < totalSeconds1)
    {
        ElapsedSeconds1++;
    }
    else
    {
        timer1.Stop();
    }
}
private void OnTimedEvent2(object sender, ElapsedEventArgs e)
{
    if (ElapsedSeconds2 < totalSeconds2)
    {
        ElapsedSeconds2++;
    }
    else
    {
        timer2.Stop();
    }
}
public void StartTimer1()
{
    if (ElapsedSeconds1 < totalSeconds1)
    {
        timer1.Start();
    }
}
public void StopTimer1()
{
    timer1.Stop();
}
public void ResetTimer1()
{
    ElapsedSeconds1 = 0;
}
public void StartTimer2()
{
    if (ElapsedSeconds2 < totalSeconds2)
    {
        timer2.Start();
    }
}
public void StopTimer2()
{
    timer2.Stop();
}
public void ResetTimer2()
{
    ElapsedSeconds2 = 0;
}
private void OnOffTrue()
{
```

```
    OnOffBool = !OnOffBool;
}
public async Task GetWeatherDataAsync(string cityName)
{
    using HttpClient client = new HttpClient();
    string url = $"{baseUrl}?q={cityName}&appid={apiKey}&units=metric";
    try
    {
        var response = await client.GetFromJsonAsync<WeatherResponse>(url);
        WeatherData = response;
        if (response != null)
        {
            Humidity = response.Main.Humidity;
            Pressure = response.Main.Pressure;
            WindSpeed = response.Wind.Speed;
        }
    }
    catch (HttpRequestException e)
    {
        Console.WriteLine($"Request error: {e.Message}");
    }
}
private async Task UpdateSensorDataAsync()        {
    using HttpClient client = new HttpClient();
    string url = $"{ESPBaseUrl}/getSensorData";
    try            {
        var response = await client.GetFromJsonAsync<SensorData>(url);
        if (response != null)                {
            CurrentSensorData = response;
        }
    }
    catch (HttpRequestException e)            {
        Console.WriteLine($"Request error: {e.Message}");
    }
}
private void IsFlame(object sender, ElapsedEventArgs e)
{
    if (CurrentSensorData != null) {
        if (CurrentSensorData.Flame == 0) {
            isFlame = 0;
            if (OnOffBool) {
                OnOffBool = false;
                preOnOffBool = true;
            }
        }
        else if (CurrentSensorData.Flame == 1)
        {
            isFlame = 1;
            if (preOnOffBool) {
                OnOffBool = true;
                preOnOffBool = false;
            }
```

```
            }
        }
    }
    private async Task SendOnOffBoolToESP()       {
        using HttpClient client = new HttpClient();
        string url = $"{ESPBaseUrl}/setOnOffBool";
        var content = new FormUrlEncodedContent(new[]          {
            new KeyValuePair<string, string>("onOffBool", OnOffBool ? "1" : "0")
        });
        try          {
            var response = await client.PostAsync(url, content);
            response.EnsureSuccessStatusCode();
        }
        catch (HttpRequestException e)          {
            Console.WriteLine($"Request error: {e.Message}");
        }
    }
    private async Task SendTemperatureThresholdToESP() {
        using HttpClient client = new HttpClient();
        string url = $"{ESPBaseUrl}/setTempCThreshold";
        var content = new FormUrlEncodedContent(new[]
        {
            new                    KeyValuePair<string,                    string>("tempCThreshold",
TemperatureThreshold.ToString())
        });
        try {
            var response = await client.PostAsync(url, content);
            response.EnsureSuccessStatusCode();
        }
        catch (HttpRequestException e)          {
            Console.WriteLine($"Request error: {e.Message}");
        }
    }
    private async Task SendHumidityThresholdToESP()       {
        using HttpClient client = new HttpClient();
        string url = $"{ESPBaseUrl}/setHumidityThreshold";
        var content = new FormUrlEncodedContent(new[]
        {
            new                    KeyValuePair<string,                    string>("humidityThreshold",
HumidityThreshold.ToString())
        });
        try          {
            var response = await client.PostAsync(url, content);
            response.EnsureSuccessStatusCode();
        }
        catch (HttpRequestException e)          {
            Console.WriteLine($"Request error: {e.Message}");
        }
    }
    private void LoadSettings() {
        // Завантажити налаштування з ізольованого сховища або іншого механізму
налаштування
```

```
            using var storage = IsolatedStorageFile.GetUserStoreForAssembly();
            using var stream = new IsolatedStorageFileStream("settings.txt",
FileMode.OpenOrCreate, storage);
            using var reader = new StreamReader(stream);
            if (int.TryParse(reader.ReadLine(), out var temperatureThreshold)) {
                TemperatureThreshold = temperatureThreshold;
            }
            if (int.TryParse(reader.ReadLine(), out var humidityThreshold)) {
                HumidityThreshold = humidityThreshold;
            }
            if (int.TryParse(reader.ReadLine(), out var eElapsedSeconds1)) {
                ElapsedSeconds1 = eElapsedSeconds1;
            }
            if (int.TryParse(reader.ReadLine(), out var eElapsedSeconds2)) {
                ElapsedSeconds2 = eElapsedSeconds1;
            }
        }
        public void SaveSettings() {
            // Зберігти налаштування в ізольованому сховищі або іншому механізмі
налаштування
            using var storage = IsolatedStorageFile.GetUserStoreForAssembly();
            using var stream = new IsolatedStorageFileStream("settings.txt", FileMode.Create,
storage);
            using var writer = new StreamWriter(stream);
            writer.WriteLine(TemperatureThreshold);
            writer.WriteLine(HumidityThreshold);
            writer.WriteLine(ElapsedSeconds1);
            writer.WriteLine(ElapsedSeconds2);
        }
    }
}
```

ДОДАТОК В

Демонстраційний матеріал у вигляді презентації

Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра КІТАР

КВАЛІФІКАЦІЙНА РОБОТА

На тему: Розробка системи автоматизації для управління якістю повітря у виробничому приміщенні

Виконав:

ст. гр. АКТАКІТ-20-1

Сириця О. О.

Керівник:

проф. каф. КІТАР

Сезонова І. К.

Мета кваліфікаційної роботи

На сьогоднішній день автоматизація управління якістю повітря є дуже актуальною у сучасному промисловому середовищі. Забезпечення високої якості повітря в промислових приміщеннях має велике значення для здоров'я та продуктивності працівників, а також для ефективності виробничих процесів.

У багатьох галузях існують строгі стандарти якості повітря, які потрібно дотримуватися з метою забезпечення безпеки працівників та якості виробничих процесів, так як певні процеси виробництва можуть бути чутливі до змін у складі повітря, вологості та температури.

Мета роботи – розробка системи автоматизації управління якістю повітря у виробничому приміщенні, з використанням датчику якості повітря.

Об’єкт розробки – процес регулювання роботи від показань з датчиків.

Предмет розробки – управління якістю повітря у виробничому приміщенні.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз існуючих проєктів;

– провести аналіз приміщень, котрим потрібна подібна система;

– провести аналіз вимог до системи управління;

– провести аналіз методів контролю якості повітря.

Фільтри повітря та їх стандарти

Таблиця 1 Порівняння класифікації фільтрів за стандартами

Категорія	Класифікація по стандартах			
	EN 779	ISO 16890	EN 1822	ISO 29463
Грубі фільтри	G1	Coarse	-	-
	G2		-	-
	G3		-	-
	G4		-	-
Середньоефективні	F5	ePM10	-	-
	F6	ePM2.5	-	-
	F7	ePM1	-	-
	F8		-	-
	F9		-	-
EPA фільтри	-	-	E10	E10
	-	-	E11	E11
	-	-	E12	E12
HEPA фільтри	-	-	H13	H13
	-	-	H14	H14

Рисунок 1 – Види фільтрів

Ефективність фільтрів та спліт -системи

Таблиця 2 Приведення старих класів до нових

Клас фільтра	ePM1	ePM2,5	ePM10	Coarse
G1	-	-	-	-
G2	-	-	-	від 30% до 50%
G3	-	-	-	від 45% до 65%
G4	-	-	-	від 60% до 85%
M5	від 5% до 35%	від 10% до 45%	від 40% до 70%	від 80% до 95%
M6	від 10% до 40%	від 20% до 50%	від 60% до 80%	більше 90%
F7	від 40% до 65%	від 65% до 75%	від 80% до 90%	більше 95%
F8	від 65% до 90%	від 75% до 95%	від 90% до 100%	більше 95%
F9	від 80% до 90%	від 85% до 95%	від 90% до 100%	більше 95%

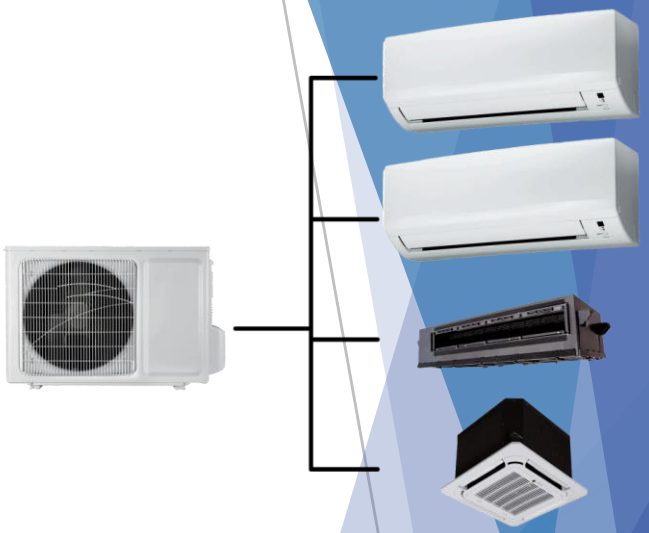

Рисунок 2 – Спліт-система

Параметри якості повітря

Таблиця 3 Стандарти концентрації TVOC

TOVC (ppb)	Вплив на здоров'я людини
від 6000	Головні болі та проблеми з нервами
від 750 до 6000	Відчуття занепокоєння та головна біль
від 50 до 750	Відчуття занепокоєння та дискомфорту
до 50	Не впливає

Таблиця 4 Розрахункова концентрація eCO2/CO2

eCO2/CO2 (ppm)	Рівень	Пропозиція
від 21500	Сильний	Забруднення повітря в приміщенні є сильним, тому потрібна вентиляція.
від 1000 до 1500	Погано	Повітря в приміщенні брудне, рекомендується вентиляція
від 800 до 1000	Звичайна якість, середній	Можна вентилювати
від 600 до 800	Добре	Немає пропозицій
від 400 до 600	Відмінно	Немає пропозицій

Основа лабораторного макету

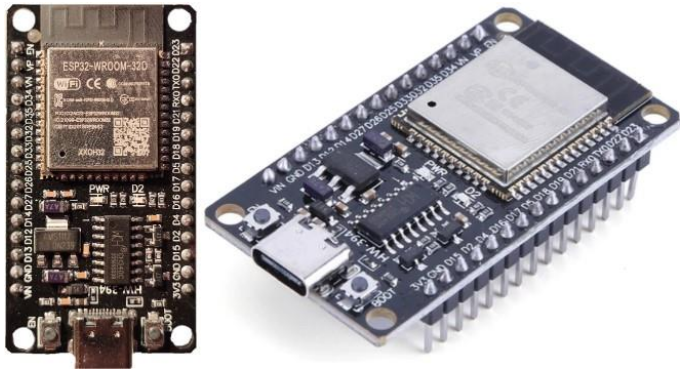

Рисунок 3 – ESP32-WROOM-32D

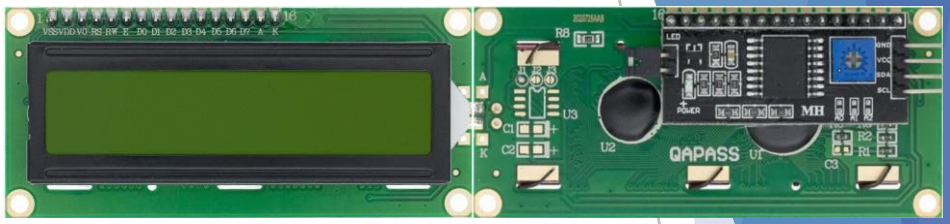

Рисунок 4 – LCD1602 I2C модуль дисплея із зеленим екраном

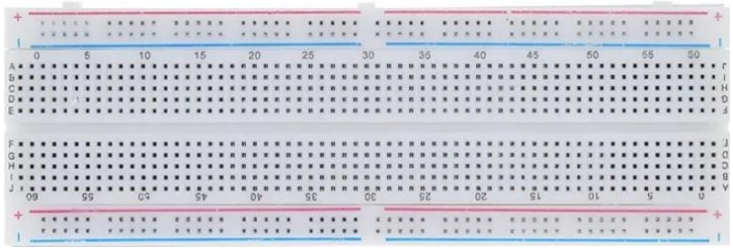

Рисунок 5 – Макетна плататипу MB-102

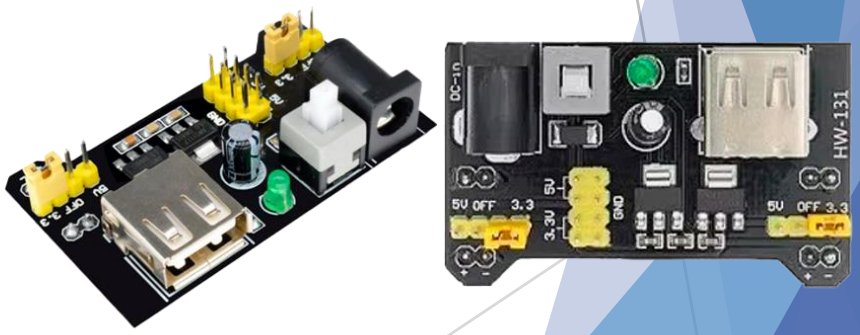

Рисунок 6 – Макетний модуль живлення типу MB-102

Датчики лабораторного макету

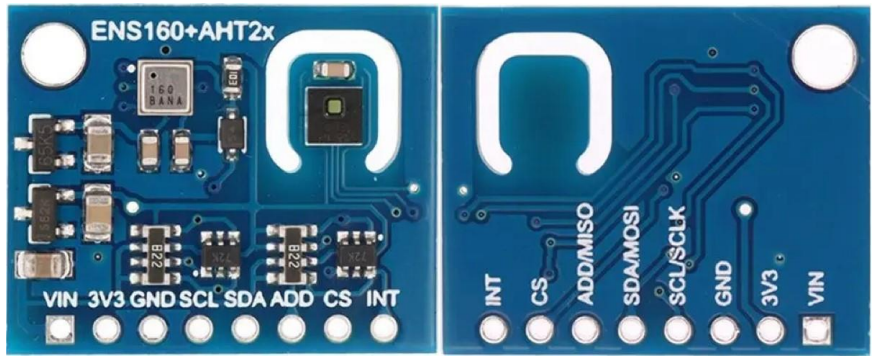

Рисунок 7 – Датчик ENS160+AHT21

Рисунок 8 – Датчик вогню

Рисунок 9 – Датчик диму MQ-2

Виконавчі елементи лабораторного макету

Рисунок 10 – ТТ Мотор-редуктор

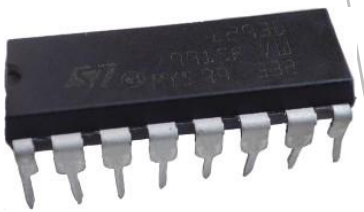

Рисунок 11 – Мікросхема драйвер моторів двоканальний L293D

Рисунок 12 – сервоприводи SG90 360 градусів

Рисунок 13 – Міні-зволожувач повітря Kinscoter DQ107

Планування та збірка макету

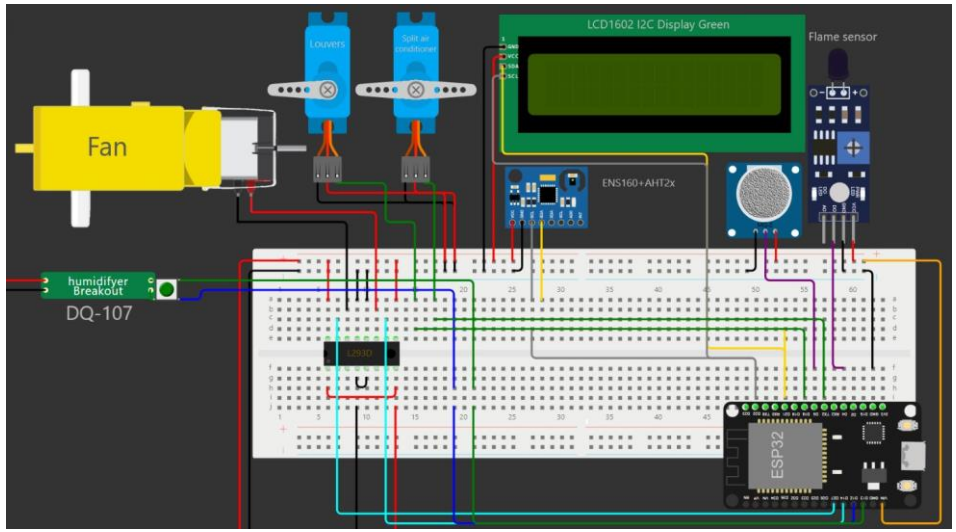

Рисунок 14 – Макет системи у віртуальному виді

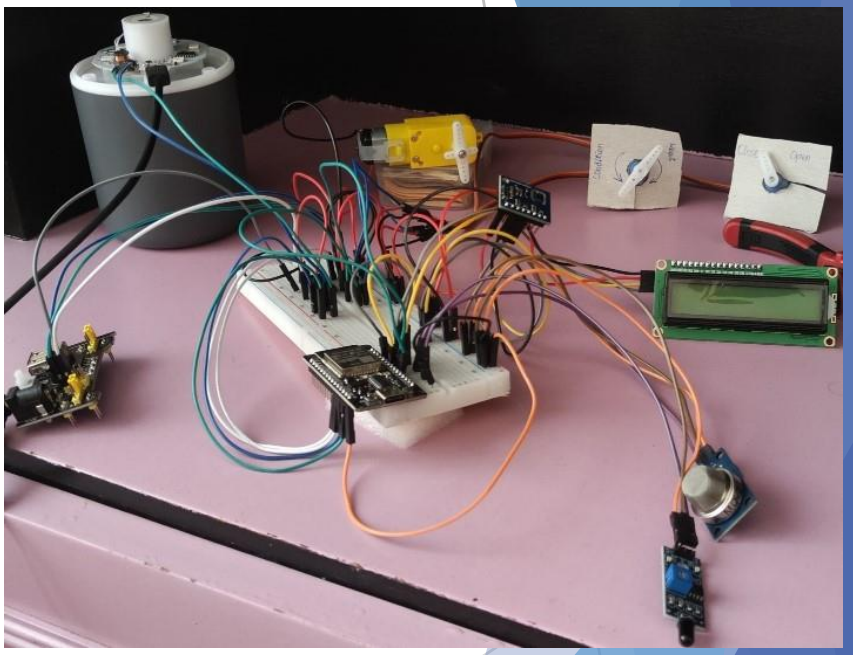

Рисунок 15 – Лабораторний макет системи

Розробка коду мікроконтролера

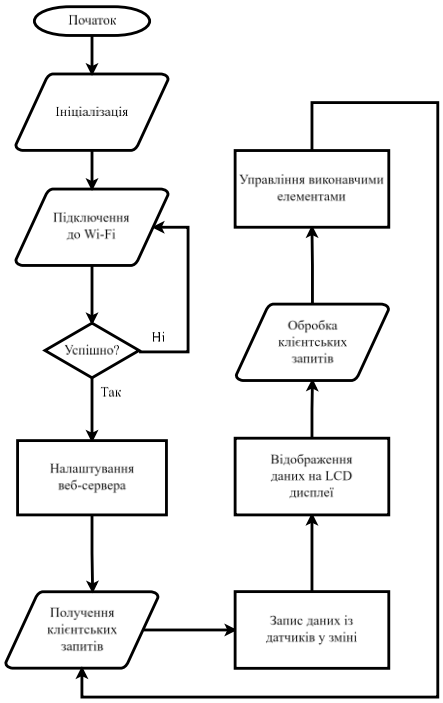

Рисунок 16 – Блок-схема алгоритму роботи мікроконтролера

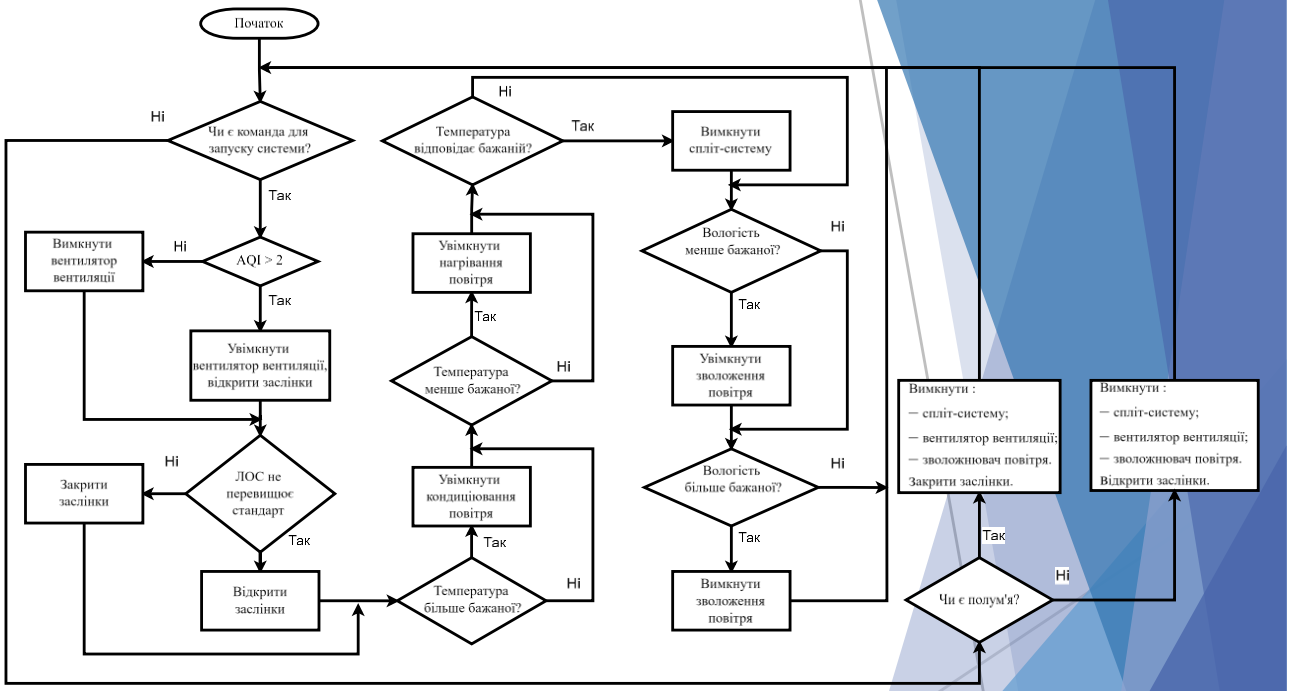

Рисунок 17 – Блок-схема управління виконавчими елементами

Розробка програми управління системою

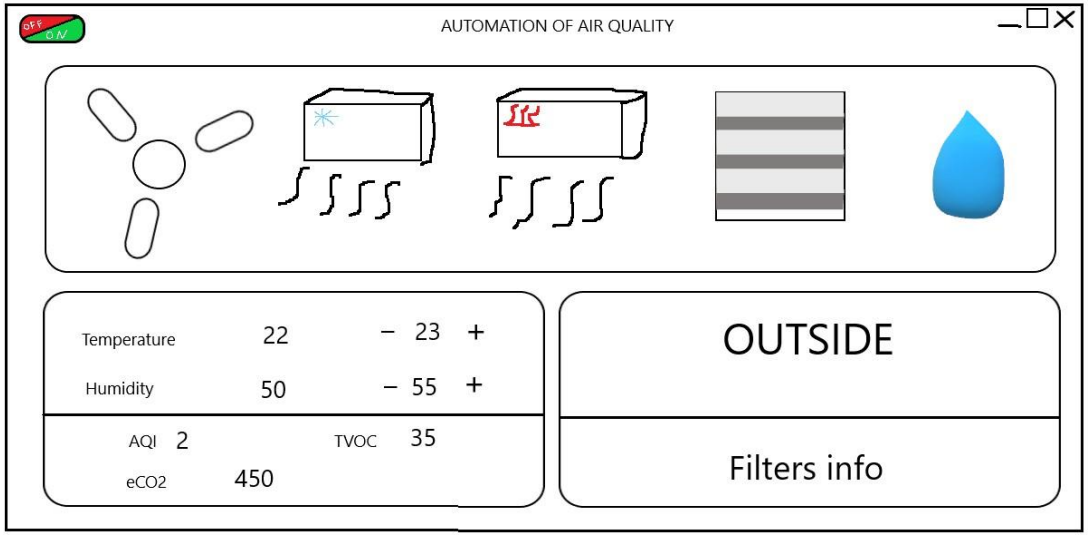

Рисунок 18 – Ескіз інтерфейсу керуючої програми

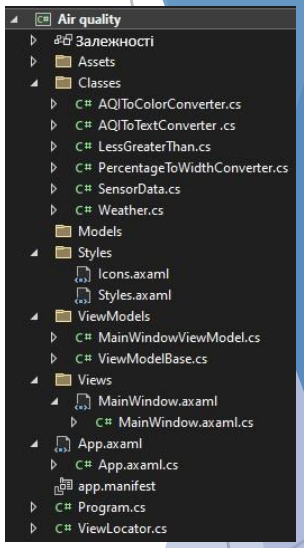

Рисунок 19 – Робочі файли, папки та класи програми

Програма управління системою автоматизації

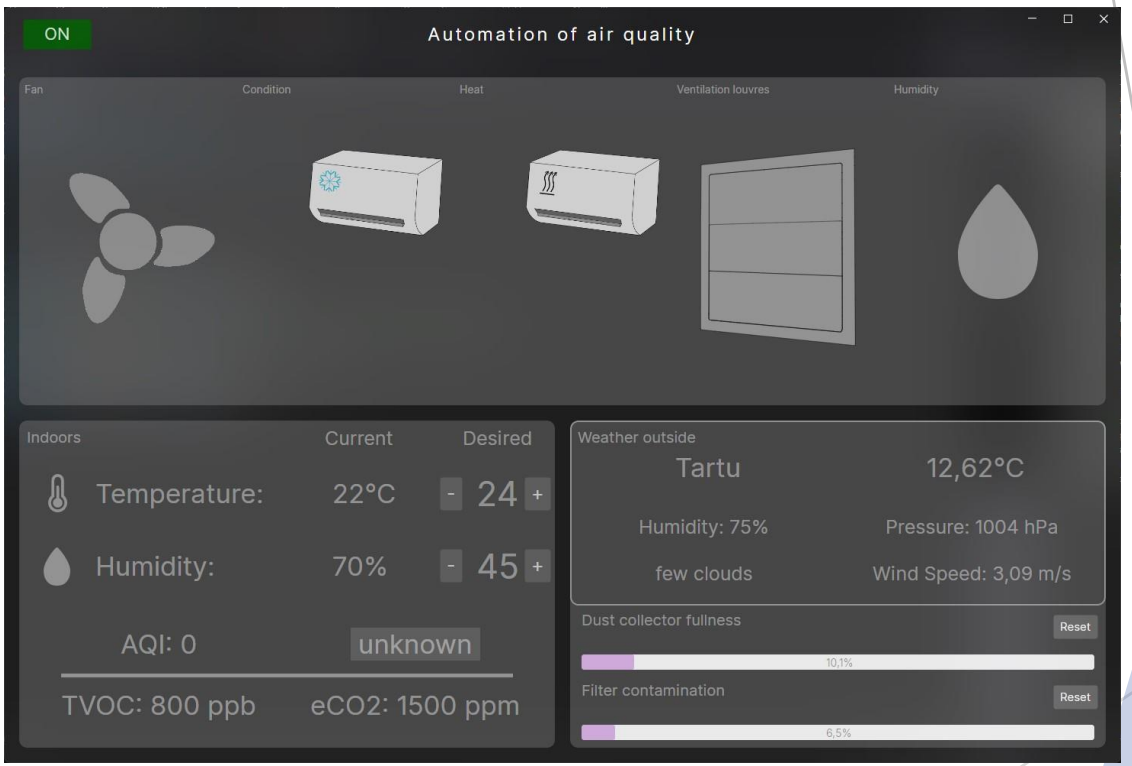

Рисунок 20 – Програма управління автоматизованою системою якості повітря

Підказки та анімовані елементи інтерфейсу

Рисунок 21 – Підказка

Рисунок 22 – Підказка при наведенні курсора на елемент програми

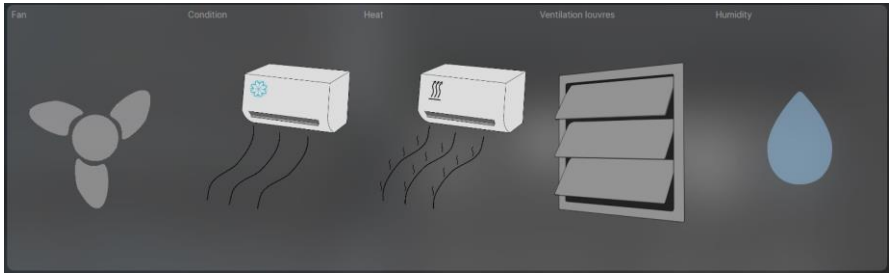

Рисунок 23 – Відображення елементів системи у роботі

Рисунок 24 – Відображення елементів системи при пожежі

Висновки

В ході виконання роботи було розглянуто та проаналізовано як саме можна контролювати якість повітря, якими методами. Було виявлено стандарти якості до повітря у приміщеннях, які фільтри використовуються для приміщень відносно того наскільки чисте повітря повинно бути.

Спроектовано та створено лабораторний макет системи автоматизації для управління якістю повітря у виробничому приміщенні. Також розроблено програмний продукт для управління системою, за допомогою якого здійснюється включення та виключення системи, а також регулювання температури і вологості повітря.

Також була розроблена модель системи автоматичного управління згідно теорії автоматичного управління, це дає змогу оптимізувати параметри регуляторів для досягнення бажаних значень показників якості повітря

Система розроблювалася для виробничих приміщень, переважно орієнтованих на вироблення електронних компонентів. Для таких приміщень є критичними параметрами статична електрика та тверді частинки. Вологість повітря контролює статичну електрику, а тверді частинки за допомогою фільтрів. Встановлювати датчики для постійного моніторингу твердих частинок немає сенсу, так як вони контролюються тільки за допомогою фільтрів у цій системі згідно стандартів якості. Розроблена система може швидко реагувати на зміни у температурі, вологості та індексу якості повітря на основі значень еквіваленту вуглекислого газу або летких органічних сполук.

ДОДАТОК Г

Відомість кваліфікаційної роботи бакалавра

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК.565100.012 ПЗ	Пояснювальна записка	А4, 72 с.
		Додаткові матеріали	
2		Демонстраційний матеріал	А4, 15 с.
		у вигляді презентації	

					ГЮИК.565100.012 ВД			
Змін.	Арк.	Номер докум.	Підпис	Дата				
Розроб.		Сириця О. О.		13.06.2024	Розробка системи автоматизації для управління якістю повітря у виробничому приміщенні	Літера	Аркуш	Аркушів
Перевір.		Сезонова І. К.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Стародубцев М.Г.						
Затв.		Невлюдов І.Ш.						