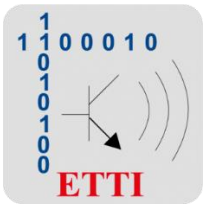
UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI
DEPARTAMENTUL INGINERIA INFORMAȚIEI



LUCRARE DE DIPLOMA

Aplicație pentru analiza calității aerului

Coordonator
Prof. Radu Hobincu

Student
Aldea Andrei

BUCUREȘTI
2022

Cuprins

Lista Figuri

Lista tabele

Sinopsis

Calitatea aerului reprezintă un factor extrem de important in calitatea vieții, putând crea probleme de sănătate atât pe termen scurt cat si pe termen lung. Pentru a putea lua deciziile si masurile de protecție corecte, este necesara o soluție care sa poată raporta starea aerului respirat.

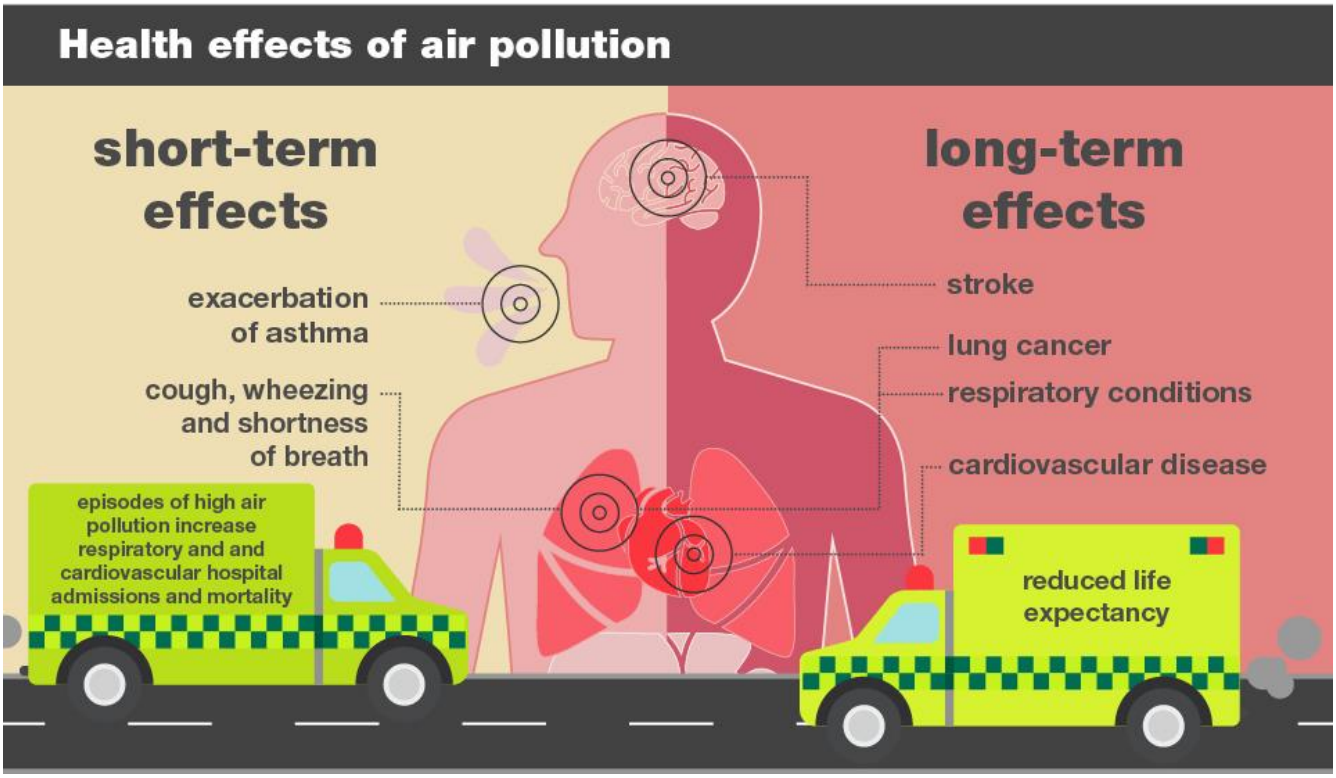
Soluția prezentata oferă utilizatorului posibilitatea de a monitoriza orașul in care se afla, dar si regiunea din jurul acestuia putând astfel urmări schimbarea calității si interacțiunea dintre orașele învecinate. Aerul si mișcările acestuia reprezintă un sistem dinamic si greu de prezis, dar folosind date deja existente putem aproxima modul in care acesta se va comporta. Un element important in calitatea aerului, pe lângă activitățile umane, îl reprezintă vântul, care poate transporta gaze si particule răspândind astfel efectul unei surse de poluare.

1. Introducere

Calitatea aerului reprezintă un factor major in influențarea stării de sănătate a populației, a florei si a faunei, indiferent de regiune sau de mediu, urban sau rural, poluarea poate avea efecte fie pe termen scurt, dar si pe termen lung, problemele cauzate putând rămâne ascunse. Din acest motiv controlul poluării asupra aerului (dar si a celorlalte medii), necesita o atenție deosebit de mare atât din partea instituțiilor de stat cat si din partea cetățenilor.

Poluarea aerului este definita ca fiind contaminarea mediului cu substanțe ce pot afecta starea de sănătate a oamenilor sau a altor organisme vii, sau care pot dauna mediului înconjurător. Chiar daca exista legi in ceea ce privește controlul poluării, aceasta sarcina se dovedește a fi una dificila, sursele de poluare fiind diverse si uneori imprevizibile, iar prognoza asupra atmosferei nefiind una perfecta. Totuși, monitorizarea aerului si a nivelului de poluare, poate fi o sarcina mai simpla, care poate oferi informații extrem de utile in prevenția apariției problemelor de sănătate. Calitatea aerului in România este reglementata prin Legea nr.104/15.06.2011 privind calitatea aerului înconjurător publicată în Monitorul Oficial al României, Partea I, nr.452 din 28 iunie 2011. [1]

Un studiu al Națiunilor Unite, arata ca pana in anul 2050, 68% din populația globului va trai in medii urbane, procentul actual fiind de 55%. Astăzi cele mai urbanizate zone sunt America de nord cu un procent de 82%, America latina cu 81% si Europa cu 74% din populație trăind la oraș. [2] In Romania gradul de urbanizare a atins in anul 2016 valoare de 57.75%. [3]



Figură 1.1

Aceste creșteri rapide ale concentrației populației in zone restrânse, mediul urban având o densitate a locuitorilor ridicata, duce la creșterea nivelului de poluare a aerului respirat de milioane de oameni care trăiesc in oraș, indiferent de activitățile desfășurate. Simpla deplasare de la domiciliu la locul de munca, expune populația la riscul dezvoltării bolilor precum alergii, boli de inima sau cancer. Chiar daca calitatea scăzută a aerului reprezintă un factor de risc pentru toata populația, unele categorii sunt vulnerabile, precum femeile însărcinate, copii, persoanele de peste 65 de ani, si cei cu boli cardiovasculare sau cancer. Adițional fata de efectul asupra sănătății, poluarea duce si la creșterea costurilor, fiind necesara alocarea unei sume suficient de mari pentru desfășurarea programelor de prevenție si control a acestor fenomene.

2. Analiza si specificarea cerințelor

In lucrarea de fata se dorește realizarea unei aplicații care sa ii permită utilizatorului sa fie informat in legătura cu starea curenta si posibilele schimbări asupra calității aerului si adițional sa ii fie prezentate si informații meteo.

Aplicația va raporta starea curenta a vremii, prezentând astfel informații precum umiditatea si temperatura aerului, viteza vântului si direcția acestuia, date preluate folosind platforma OpenWeather, care pune la dispoziție un API public.

Similar vor fi prezentate date despre starea curenta a calității aerului, calculându-se AQI-ul si prezentând concentrațiile mai multor poluanți precum:

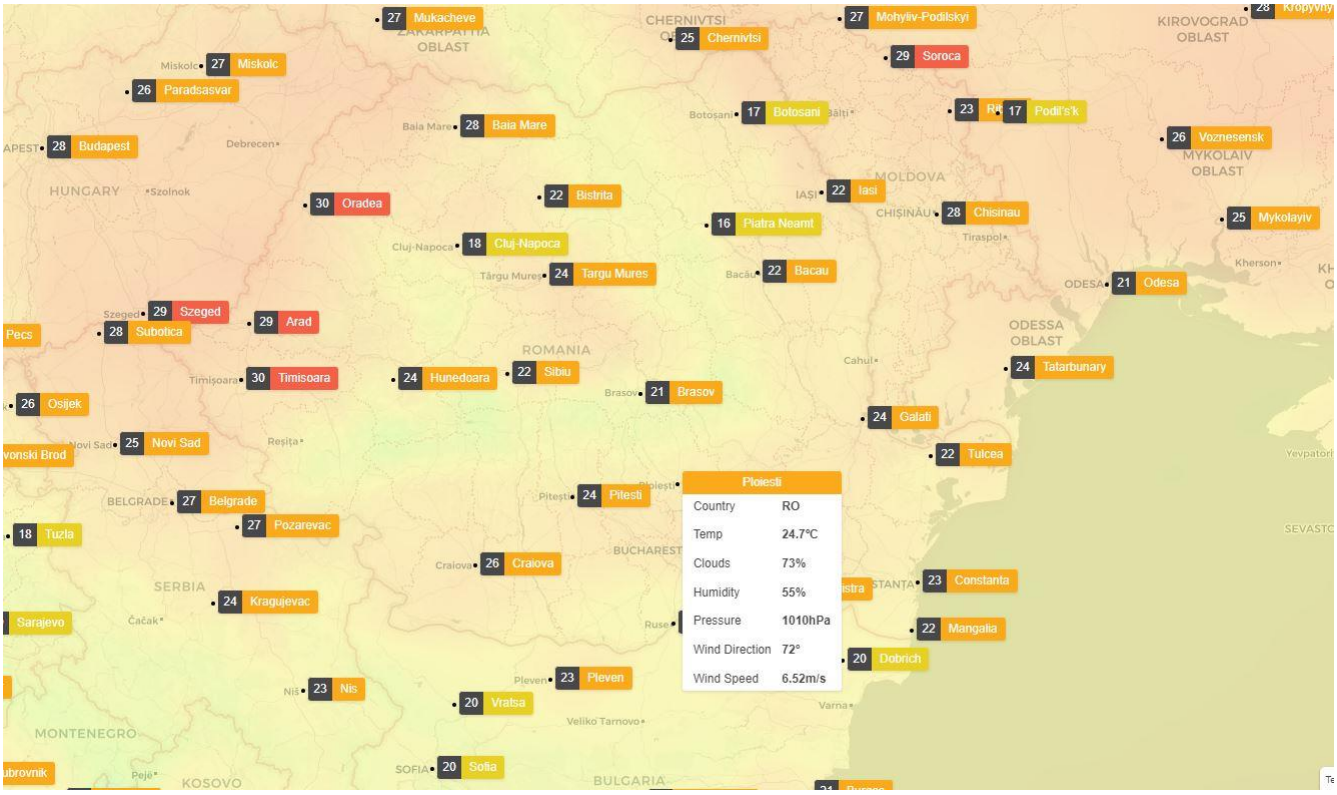
- Monoxidul de carbon (CO), un gaz incolor, inodor si insipid, emanat de arderea combustibililor folosiți in diverse dispozitive care se regăsesc in casele oamenilor;
- Dioxid de sulf (SO2), emanat la arderea combustibililor, acesta reacționând cu alți compuși din aer produce particule mici care contribuie la concentrația de PM;
- Particulele de mici dimensiuni (PM) reprezintă un mix complex de particule solide si lichide, emanate in principal prin combustia realizata de vehicule, zone industriale sau cea de natura domestica. Clasificarea acestora se face in trei categorii: PM10 – particule (aspre) cu dimensiuni de pana la 10 microni, PM2.5 – particule (fine) cu dimensiuni de pana la 2.5 microni si PM0.1 – particule (ultra fine) cu dimensiuni de pana la 0.1 microni.
- Dioxid de azot (NO2) un gaz produs adesea împreuna cu oxidul de azot (NO).
- Amoniac (NH3), emanat fie din surse naturale sau făcute de om, gazul contribuie la concentrația de PM datorita reacțiilor chimice care duc la producerea de acid sulfuric si acid nitric.
- Ozon (O3), care spre deosebire de restul gazelor nu este emis in mod direct ci apare prin reacții fotochimice cu gaze precum NO sau NO2. Efectele produse de acest gaz sunt adesea pe termen scurt, efectele pe termen lung precum bolile cardiovasculare nefiind susținute de foarte multe dovezi.

Având in vedere multitudinea de gaze luate in considerare si diferențele intre acestea, predicția in ceea ce privește modificările concentrațiilor acestora vor fi realizate folosind modele diferite, căutat astfel obținerea unui rezultat cat mai apropiat de realitate.

3. Soluții existente

OpenWeather

Similar cu lucrarea de fata, exista sisteme care pun la dispoziție date despre starea vremii si despre calitatea aerului. Platforma OpenWeather, a aerului API a fost folosit pentru obținerea datelor, are propria pagina web unde prezinta aceste informații. Aceștia prezinta datele afișate pe o harta care își schimba culoare in funcție de parametrul pe care utilizatorul își dorește sa îl vizualizeze: temperatura, presiune, viteza vântului, nori, precipitații. Un exemplu de harta care prezinta date despre temperatura se poate vedea mai jos.

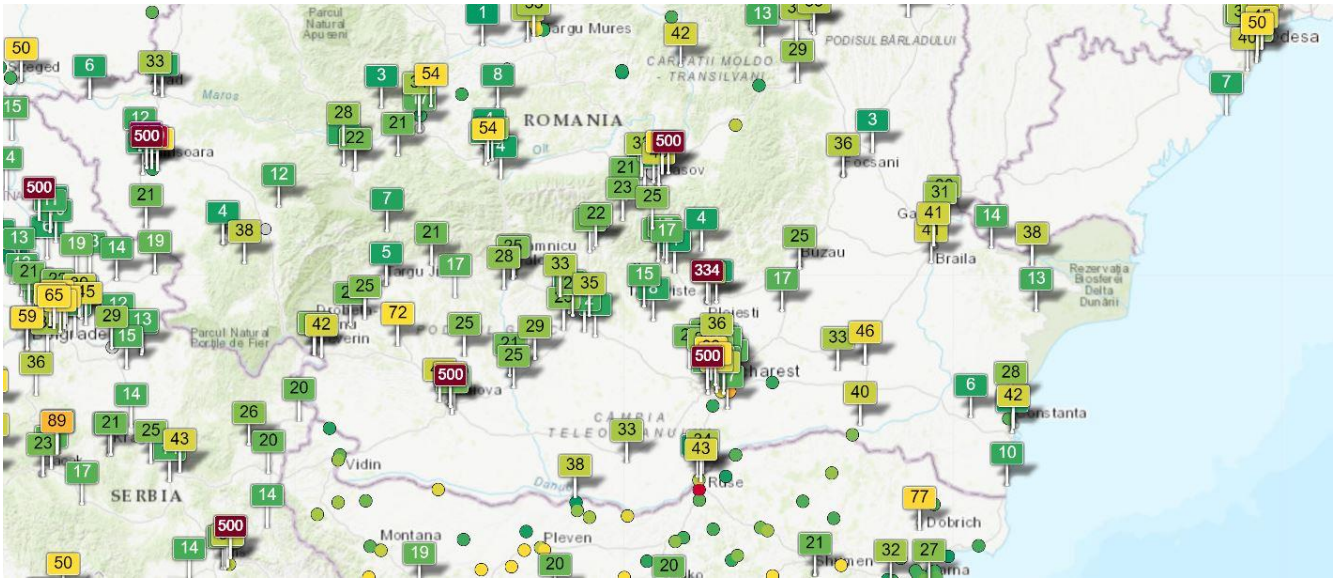


Figură 3.1

Platforma nu dispune si de o reprezentare vizuala a datelor legate de calitatea aerului.

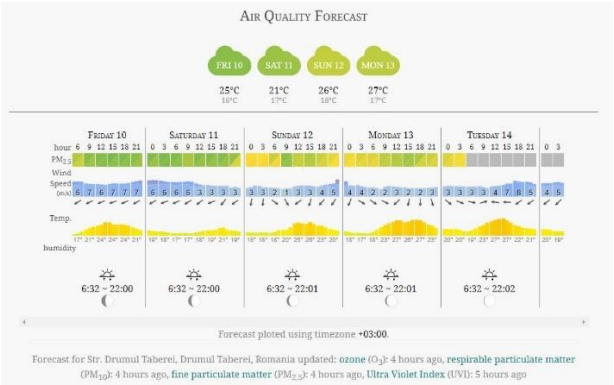
AQICN

O prezentare vizuala mai buna este oferita de platforma Aqicn, care marchează pe harta poziția si valoarea calității aerului înregistrata de o anumita stație locala. Aceștia oferă si o predicție a acestor valori pentru câteva zile.



Figură 3.2

O prezentare a predicției calității poate fi observata mai jos.



Figură 3.3

Putem observa ca este prezentata o valoare estimativa a poluantului PM2.5 pentru următoarele 4 zile.

4. Soluția propusa

4.1. Tehnologii folosite

In realizarea acestui proiect au fost folosite mai multe tehnologii si medii de dezvoltare. Fiind folosite mai multe limbaje de programare (Java, Python, C), pentru fiecare a fost necesar alt mediu de dezvoltare. Pentru realizarea codului principal al aplicației, dezvoltata cu limbajul Java, a fost folosit mediul de dezvoltare IntelliJ IDEA dezvoltat de JetBrains. Pentru antrenarea modelului de regresie, folosit in predicția schimbării cantităților de poluanți, a fost utilizat Spyder, un mediu de dezvoltare open-source care vine cu mai multe pachete utile instalate precum: NumPy, SciPy, Matplotlib, pandas, etc. Codul utilizat in controlul senzorilor cu ajutorul plăcii Arduino Uno a fost scris si încărcat pe microcontroler folosind Arduino IDE. Alte tehnologii necesare in realizarea proiectului vor fi descrise mai detaliat in cele ce urmează.

4.1.1. OpenWeather API

OpenWeather este o platforma ce pune la dispoziție o multitudine de unelte utile, are pot fi folosite pentru monitorizarea calității aerului dar nu numai. Platforma oferă de asemenea informații despre vreme, starea drumurilor, alerte meteo, intensitatea UV, etc. Platforma oferă date despre peste 200.000 orașe, acestea fiind colectate si procesate din mai multe surse precum sateliți, radar si spatii meteo locale.

In aceasta lucrare sunt folosite doua API-uri: unul de la care colectam date despre stațiile meteo dintr-o anumita regiune, si un alt API care oferă informații despre calitatea aerului la o anumita stație. Utilizarea detaliata a acestor servicii va fi detaliata in capitolul 4.2.

4.1.2. Java, JavaFX

Java

Aplicația este realizata folosind limbajul de programare Java, un limbaj dezvoltat la începutul anilor `90 de către Sun Microsystems. Limbajul este unul puternic tipizat, orientat pe obiecte. La descărcarea Java utilizatorul va primi Java Runtime Environment (JRE). Acest pachet conține Java Virtual Machine (JVM), clasele de baza si bibliotecile care pot fi folosite, fiind tot ceea ce este necesar pentru a rula aplicații Java. [4]

JRE nu vine si cu uneltele necesare creării de aplicații, unelte care fac parte din ava Development Kit (JDK). Acest pachet conține resurse necesare dezvoltării aplicațiilor precum o mașina virtuala privata (JVM), un compilator (javac), un arhivator (jar), un generator de documentație (javadoc), etc. [5]

JavaFX

Interfața aplicației este construita folosind JavaFX, un set de unelte complet, folosit pentru dezvoltarea aplicațiilor complexe. Aspectul aplicațiilor poate fi modificat si adaptat ușor la cerințele proiectului folosind Cascading Style Sheets (CSS) pentru controlul stilului elementelor de UI.

JavaFX 2.2 si versiunile mai noi sunt integrate complet in Java SE 7 Runtime Environment (JRE) si in Java Development Kit (JDK). Pentru ca JDK-ul este disponibil pentru toate platformele desktop ((Windows, Mac OS X, and Linux), aplicațiile care folosesc JavaFX sunt compatibile pe mai multe platforme. Oracle asigura lansarea sincronizata a update-urilor pentru toate platformele si oferă un program de mentenanța pentru companiile care folosesc JavaFX. [6]

4.1.3. Python

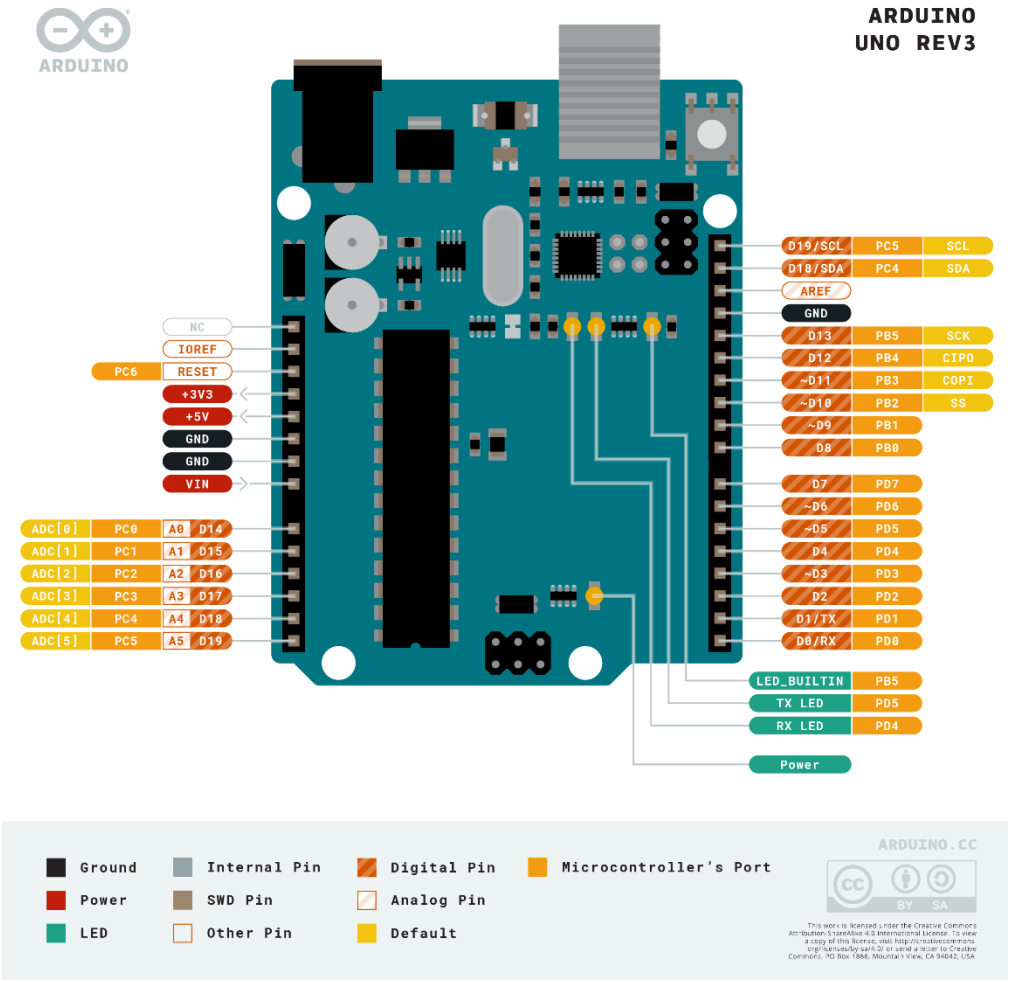
Aplicația prezinta si opțiunea de a face o predicție asupra poluanților din atmosfera. Pentru ca aceasta funcționalitate sa fie posibila, a fost necesara antrenarea unui model de regresie polinomiala, realizat cu ajutorul limbajului de programare Python si a bibliotecii Scikit-learn.

Python este un limbaj de programare dinamic, multi-paradigma, creat in 1989 de programatorul olandez Guido van Rossum. Limbajul pune accentul pe simplitatea si aspectul curat al codului, oferind posibilitatea de a exprima o idee intr-un mod clar si concis. Limbajul poate fi folosit si ca limbaj orientat pe obiecte, dar permite si programarea imperativa, funcționala sau procedurala. [7]

Scikit-learn este o biblioteca open-source care oferă algoritmi ce pot fi folosiți in învățarea supervizata si nesupervizata, procesarea datelor, si evaluarea rezultatelor. In acest proiect a fost folosit un algoritm de regresie polinomiala pentru predicția cantității diverșilor poluanți din atmosfera in următoarea ora, folosind datele despre concentrațiile curente si viteza vântului. [8]

4.1.4. Arduino si senzorii utilizați

Pentru colectarea datelor despre calitatea aerului si alte date meteo cum ar fi temperatura, umiditate si presiune, a fost folosit placa de dezvoltare Arduino UNO împreuna cu senzorii necesari. Arduino UNO are la baza microcontrolerul ATmega328P, care dispune de 14 pini digitali (intrare\ieșire), 6 intrări analogice, având frecventa reglata de un rezonator ceramic de 16MHz. Tensiunea oferita de pini este de 5V, iar curentul de 20 mA, alimentarea plăcii fiind posibila prin conectorul USB sau de la o alta sursa externa care este acceptata de regulatorul de tensiune inclus, intre 5V si 12V. [9]



Figură 4.1

In colectarea datelor au fost folosiți următorii senzori: MQ-137, MQ-5, BMP280, THD11.

MQ-137 si MQ-5

Pentru detecția gazelor din aer sunt folosiți senzorii din seria MQ care oferă o soluție low-cost si fiabila. Senzorii au in componenta lor un element rezistiv care se încălzește si un senzor electrochimic. Elementul de încălzire este necesar pentru a aduce suprafața reactiva a senzorului, de obicei un oxid metalic, la o temperatura ideala detecției. La reacția cu gazul rezistenta senzorului variază, aceasta variație fiind dependenta de concentrația gazului din atmosfera, poate fi măsurata si astfel determinata concentrația.

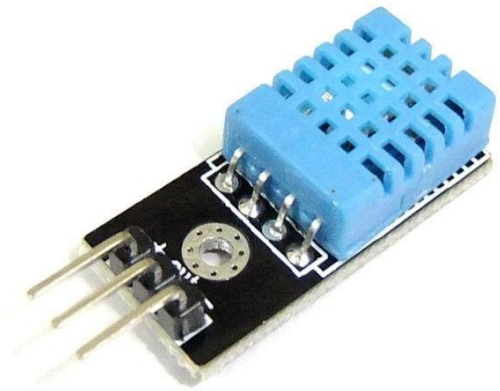
Figură 4.2

Senzorii MQ trebuie folosiți in medii închise si la temperaturi care sa nu fie prea mari sau prea mici. Pentru ca elementul de încălzire este ținut la o temperatura constanta fără a se tine cont de temperatura ambientala, daca este folosit in medii cu temperaturi prea mari sau prea mici, măsurătorile pot fi eronate. Pentru ca măsurătorile sa fie cat mai corecte, un timp de preîncălzire este necesar. Acest timp variază in funcție de modelul senzorului. [10]

DHT11

Senzorul DHT11 este folosit pentru măsurarea temperaturii si a umidității. Acesta are inclus un microcontroler pe 8bit care este folosit in transmiterea seriala a datelor măsurate, fiind deja calibrat este ușor de folosit, in cazul de fata, cu un microcontroler precum Arduino. Senzorul poate măsura temperaturi intre 0°C si 50°C, si umiditate intre 20% si 90% cu o acuratețe de ±1°C si ±1%. Deoarece este un senzor cu un preț redus acuratețea datelor nu va fi foarte ridicata, iar datele de la senzor pot fi citite la fiecare 2 secunde, spre deosebire de DHT22, dar care are dimensiuni mai mari si este mai scump.

Pentru măsurarea umidității si temperaturii, este folosit un senzor capacitiv de umiditate si un termistor pentru măsurarea temperaturii. Este important de reținut faptul ca DHT11 măsoară umiditatea relativa, reprezentând cantitatea de vapori din aer in comparație cu starea sa de saturație, aerul saturat având o umiditate relativa de 100%, iar cel uscat de 0%. Punctul de saturație se schimba cu temperatura aerului, aerul rece putând conține mai putini vapori de apa decât cel cald. [11]



Figură 4.3

	Condiții	**Minim**	**Recomandat**	**Maxim**
Tensiunea de alimentare		3V DC	5V DC	5.5V DC
Curent	Măsurare	0.5 mA		2.5 mA
	Mediu	0.2 mA		1 mA
	Standby	100 uA		150 uA
Perioada de măsurare		1 secunda		

Tabel 4.1

BMP280

Pentru măsurarea presiunii atmosferice se folosește senzorul BMP280, un senzor compact, de mici dimensiuni, si cu un consum scăzut. Senzorul poate măsura presiunea si temperatura cu o acuratețe buna, iar pentru ca presiunea atmosferica se schimba in funcție de înălțimea la care este efectuata măsurătoarea, senzorul poate fi folosit si ca altimetru. [12]

Comunicarea cu Arduino se realizează prin protocolul I2C.

4.2. Aplicația

Prezentarea generala

Aplicația realizata oferă un mediu in care datele meteo si cele legate de calitatea aerului pot fi analizate si observate într-o interfața simpla si ușor de folosit. In centrul interfeței se afla un panou unde acestea sunt prezentate vizual, stațiile meteo fiind reprezentate de cercuri colorate in funcție de valoarea AQI-ului calculat, distribuția lor respectând așezarea pe harta. Fiecare stație meteo poate fi analizata in detaliu, într-o fereastra ce se deschide separat de cea principala, unde sunt afișate: temperatura, umiditatea, viteza si direcția vântului, cantitatea poluanților la momentul actual si o predicție a acestor valori in următoarea ora si o lista cu posibilele surse de poluare. Aplicația prezinta si un meniu de ajutor in care utilizatorul poate găsi informai relevante despre modul de operare al funcțiilor de baza.

Pe lângă aceste funcționalități de baza, meniurile suplimentare oferă posibilitatea de a folosi aplicația pentru salvarea automata a datelor la un interval fix de o ora, salvarea manuala si încărcarea acestora din memorie, dar si o interfață pentru a observa datele colectate de senzorii conectați la placa arduino.

Folosind un API public, care necesita o cheie de acces, aplicația oferă posibilitatea de a schimba cheia de baza salvata in memorie, utilizatorul putând astfel folosi un plan diferit de cel gratuit pus la dispoziție de platforma OpenWeather. Planul de baza folosit in mod implicit are câteva limitări, cum ar fi un număr de doar 60 apeluri pe ora si o disponibilitate a serviciilor de 95% din timp.

Colectarea datelor

Pentru a putea analiza starea aerului din zona dorita, este necesara cunoașterea coordonatelor, care trebuie completate in căsuța de text cu numele „Coordinates”. Coordonatele trebuie introduse in formatul corect, folosit la efectuarea cererii către API, si anume: longitudine stânga, latitudine jos, longitudine dreapta, latitudine sus, separate prin virgula, aceste valori formând un dreptunghi care delimitează zona geografica in care vor fi căutate stații meteorologice. Putând fi introduse orice valori, este necesar un pas de validare care va asigura corectitudinea datelor introduse. Se vor verifica astfel intervalele din care fac parte coordonatele, latitudinea putând fi in intervalul [-90, 90], iar longitudinea in intervalul [-180, 180], dar si faptul ca suprafața determinata de aceste valori nu depășește 5 deg^2, aceasta fiind o limitare a serviciului pus la dispoziție de platforma OpenWeather. Înainte de validarea datelor se efectuează preprocesarea lor, eliminând spatiile nedorite si separând valorile folosind delimitatorul virgula. Orice abatere de la formatul corect va duce la neefectuarea cererii către platforma si generarea unui mesaj de eroare care va informa utilizatorul cu privire la greșeala făcută. De exemplu introducerea a mai puțin sau mai mult de patru valori va genera mesajul de eroare „Coordiates are not complete !”, afișat într-o fereastra noua.

Pentru a putea testa funcționalitatea aplicației mai ușor, in cazul in care nu se completează coordonatele dorite, se vor folosi valorile implicite: 24.41, 44.17, 26.90, 45.95, delimitând astfel zona de interes ca fiind una ce cuprinde localități precum Brașov, București, Făgăraș, etc.

După introducerea corecta a coordonatelor este necesara apăsarea butonului „Get Data”, care va duce la efectuarea unei serii de operații necesare pentru a afișa datele dorite.

In primul rând va fi apelat API-ul platformei OpenWeather pentru a primi datele legate e stațiile meteorologice din zona dorita. Datele vor avea in componenta lor o lista de stații meteorologice despre care se cunosc numele si coordonatele. Coordonatele sunt folosite pentru a efectua un alt apel, de data aceasta către un alt serviciu ce va oferi datele de interes si anume cele despre concentrațiile poluanților si starea vremii, ce vor fi stocate in clasa „Station”, o lista de astfel de clase fiind la rândul ei stocata in clasa „StationMap”.

Afișarea datelor

In al doilea rând se va realiza procesul de afișare pe ecran al stațiilor, acestea fiind reprezentate de un cerc colorat diferit in funcție de calitatea aerului raportata in acea zona si de o linie neagra care va avea lungimea si orientarea diferita in funcție de viteza respectiv direcția vântului. Lungimea liniilor este relativa pentru fiecare set de date, acestea fiind normate la cea mai mare viteza găsita in momentul colectării datelor, cea mai lunga linie pe ecran având maxim 40 pixeli. Pentru a putea face acest lucru, prima data vor fi șterse de pe ecran datele afișate anterior, in cazul in care acestea exista, apoi pentru ca spațiul de afișare este un pătrat cu latura e 400 px, iar poziția stațiilor este data de coordonate sferice, trebuie folosita o funcție de conversie a poziției.

```
public Point2D convertToPoint2DInBound(CoordBoundingBox boundingBox)
{
    double OFFSET_WIDTH  = 0.1 * Config.DATA_PANE_WIDTH;
    double OFFSET_HEIGHT = 0.1 * Config.DATA_PANE_HEIGHT;

    double distX = Math.abs(lon - boundingBox.getMinLon());
    double distY = Math.abs(lat - boundingBox.getMinLat());

    double maxDistX = Math.abs(boundingBox.getMaxLon() - boundingBox.getMinLon());
    double maxDistY = Math.abs(boundingBox.getMaxLat() - boundingBox.getMinLat());

    double distXNormalized = distX / maxDistX;
    double distYNormalized = distY / maxDistY;
```

```
        distYNormalized = 1.0f - distYNormalized;

        Point2D point = new Point2D(
                OFFSET_WIDTH  + distXNormalized * (Config.DATA_PANE_WIDTH  - 2.0f * OFFSET_WIDTH),
                OFFSET_HEIGHT + distYNormalized * (Config.DATA_PANE_HEIGHT - 2.0f * OFFSET_HEIGHT)
        );

        return point;
    }
```

Funcția prezentata mai sus realizează conversia unui punct din coordonate sferice in coordonate carteziene, adăugând o spațiere fata de marginile chenarului egala cu 10% din mărimea acestuia, fără aceasta spațiere jumătate din cercul care reprezintă o stație plasata pe marginea chenarului nu ar fi vizibil.

Figură 4.4

In figura de mai sus se pot observa culorile diferite date de calitatea aerului in jurul stației respective si liniile folosite pentru reprezentarea vântului.

Procesarea datelor

După colectarea si afișarea datelor, pentru a realiza o analiza a acestora este necesara apăsarea butonului „Calculate”, care va efectua apelul funcției „calculate”. Aceasta, pe lângă validările necesare, va efectua trei operații: găsirea posibilelor surse de poluare, aproximarea modificării cantităților de poluanți in următoarea ora si sortarea in funcție de nivelul de poluare a posibilelor surse.

```
public static void calculate()
    {
        areCalculationDone = true;


        System.out.println("Calculating...");


        if(stationsMap == null) {
            System.out.println("No data to calculate: null stationsMap");
            return;
        }
        if(stationsMap.getStations() == null) {
            System.out.println("No data to calculate: null stationsMap.getStations()");
            return;
        }


        for(Station station : stationsMap.getStations()){
            station.findWindSource(stationsMap);
            station.componentChange();
            station.sortPossibleSources();
        }
    }
```

Prima funcție apelata, „findWindSource” are scopul de a găsi stațiile spre care sufla vântul de la stația actuala. Datorita dimensiunilor scăzute, particulele si moleculele de gaz ce constituie poluanții pot fi transportați de vânt pe distante care variază intre zeci de metri si mii de kilometri, dar pentru aplicația de fata se vor lua in considerare doar stațiile aflate pe o raza de 50Km, crescând astfel șansa ca poluarea măsurata la stația respectiva sa vie influențata de stația de unde se efectuează căutarea. Pentru a găsi aceste stații trebuie determinat prima data daca direcția vântului este in treptata către o alta stație, pentru asta se va folosi o funcție ajutătoare denumita „lookAt” care primește ca parametrii cele doua puncte si

unghiul dat de direcția vântului. Funcția va întoarce un număr intre 0 si 180, reprezentând unghiul format intre o dreapta trasata intre cele doua puncte folosite si o dreapta determinata de direcția vântului.

După găsirea posibilelor surse de poluare, pentru a le putea afișa într-o ordine relevanta, acestea vor fi sortate in funcție de valoarea AQI-ului, mărime ce indica gradul de poluare. Pentru calculul acestei valori se folosesc concentrațiile poluanților înregistrați, si punctele critice ale acestor concentrații, așa cum este descris mai jos.

$$I = \frac{I_{high} - I_{low}}{C_{high} - C_{low}} (C - C_{low}) + I_{low}$$

Unde:

I - indexul pentru calitatea aerului

C - concentrația de poluant

C_{low} - punctul critic al concentratiei ≤ C

C_{high} - punctul critic al concentratiei ≥ C

I_{low} - punctul critic al indexului corespunzător lui C_{low}

I_{high} - punctul critic al indexului corespunzător lui C_{high}

Valorile punctelor critice pot fi regăsite in tabelul de mai jos, oferit de Biroul pentru Calitatea Aerului (US EPA) [13]

O3 (ppb)	PM2.5 (µg/m3)	PM10 (µg/m3)	CO (ppm)	SO2 (ppb)	NO2 (ppb)	AQI	AQI
Clow – Chigh (avg)	Clow – Chigh (avg)	Clow – Chigh (avg)	Clow – Chigh (avg)	Clow – Chigh (avg)	Clow – Chigh (avg)	Ilow – Ihigh	Categorie
-	0.0-12.0	0-54	0.0-4.4	0-35	0-53	0-50	Bun
-	12.1-35.4	55-154	4.5-9.4	36-75	54-100	51-100	Moderat
125-164	35.5-55.4	155-254	9.5-12.4	76-185	101-360	101-150	Nesănătos pentru grupurile sensibile
165-204	55.5-150.4	255-354	12.5-15.4	186-304	361-649	151-200	Nesănătos
205-404	150.5-250.4	355-424	15.5-30.4	305-604	650-1249	201-300	Foarte nesănătos
405-504	250.5-350.4	425-504	30.5-40.4	605-804	1250-1649	301-400	Periculos
505-604	350.5-500.4	505-604	40.5-50.4	805-1004	1650-2049	401-500	

Tabel 4.2

Daca sunt măsurați mai mulți poluanți, pentru a determina AQI-ul se va calcula valoarea I pentru fiecare poluant si se va alege maximul dintre aceste valori. In proiectul prezentat se va calcula valoarea I pentru 5 poluanți, si anume: Pm 2.5, Pm 10, CO, SO2, NO2. Această valoare se va afișa într-o interfață separata, dedicata stației pentru care a fost calculata. Pentru realizarea acestei operații se va folosi o funcție speciala.

```
public static int calculateAQI(Components components)
    {
        int aqi[] = {
                pm2_5AQI(components.getPm2_5()),
                pm10AQI(components.getPm10()),
                coAQI(components.getCo()),
                so2AQI(components.getSo2()),
                no2AQI(components.getNo2())
        };
```

```
        Arrays.sort(aqi);

        return aqi[4];
    }
```

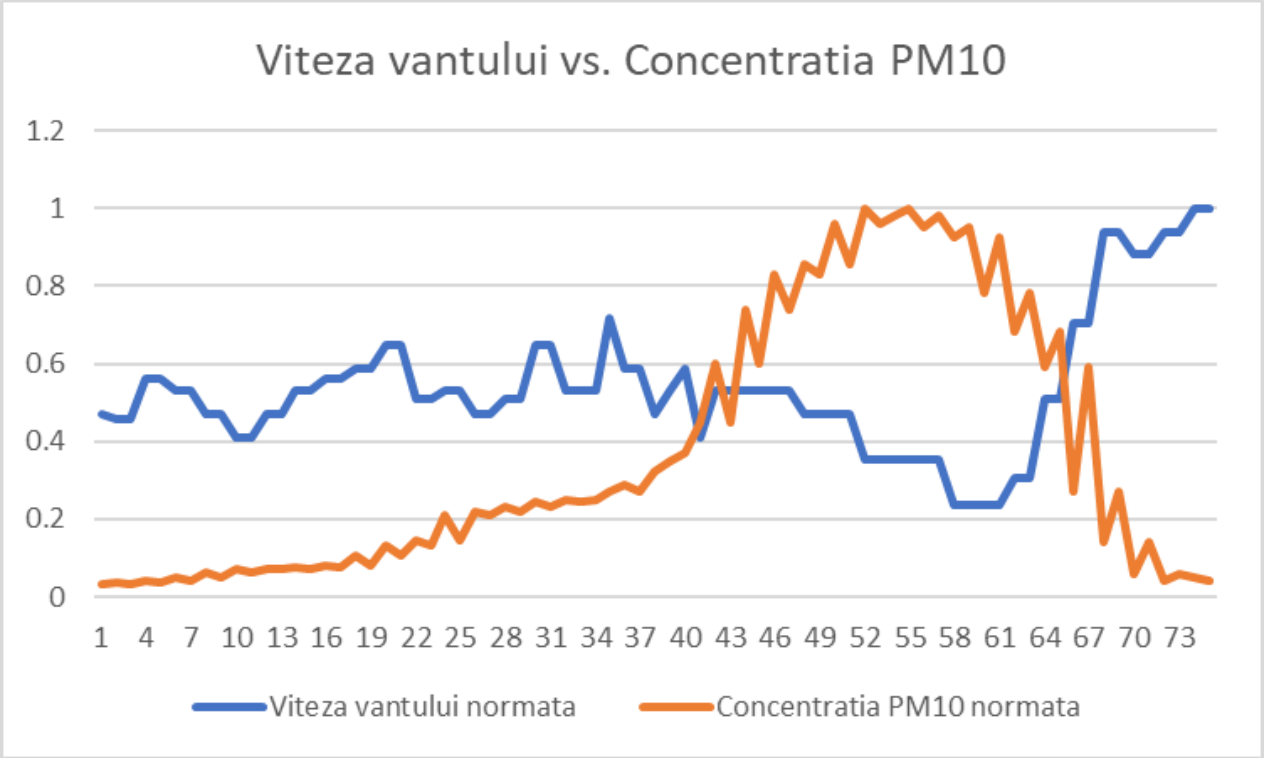
Funcția prezentata mai sus, realizează calculul valorii AQI maxime, calculând pentru fiecare poluant valoarea I. Aceste valori sunt stocate intr-un vector care este sortat pentru a obține la sfârșitul lui cel mai mare element. Aceasta operație este rapida, algoritmul de sortare implementat in Java fiind unul rapid (cu o complexitate de timp de O(n log n)), dar si pentru ca sunt foarte puține elemente.

Pentru a putea realiza o predicție asupra concentrațiilor poluanților in următoarea ora se va folosi un model de regresie polinomiala. Datele necesare antrenării modelului, deoarece nu au fost puse la dispoziție de platforma OpenWeather in mod gratuit, a fost necesara implementarea unui mod de colectare automata. In meniul „Tools" al interfeței principale poate fi regăsit butonul „Start auto data collect" care va duce la pornire procesului de colectare a datelor. Programul va porni o funcție pe un thread separat de cel principal, unde la un interval de o ora va salva datele primite prin apelul API-ului, intr-un fișier de tipul JSON, cu numele de tipul: „map_file_06_06_2022_18_12_43.json", reprezentând data si ora la care a fost salvat fișierul.

La închiderea procesului de colectare aplicația va genera un fișier text in care vor fi notate câteva statistici despre acesta. De exemplu, in procesul de colectare a datelor folosite in aplicația prezentata fișierul generat a conținut datele de mai jos.

```
AppMetrics:
  totalAPICalls = 1501
  totalFilesSaved = 79
  startTime = Wed May 25 22:51:27 EEST 2022
  endTime = Sun May 29 05:44:24 EEST 2022
  totalHoursRunning = 78.0
```

După cum poate fi observat, datele au fost adunate pe parcursul a 78 de ore, începând la ora 22:51 pe data de 25 Mai si a fost oprit pe data de 29 Mai la ora 05:44. Au fost efectuate 1501 apeluri ale serviciului si au fost salvate 79 de fișiere. După procesarea acestor fișiere au fost obținute 1279 puncte ce pot fi folosite in antrenarea modelului. Datele relevante au fost viteza vântului si concentrația poluanților, aceste doua valori având o relație invers proporționala, când viteza vântului creste, concentrația poluanților scade.



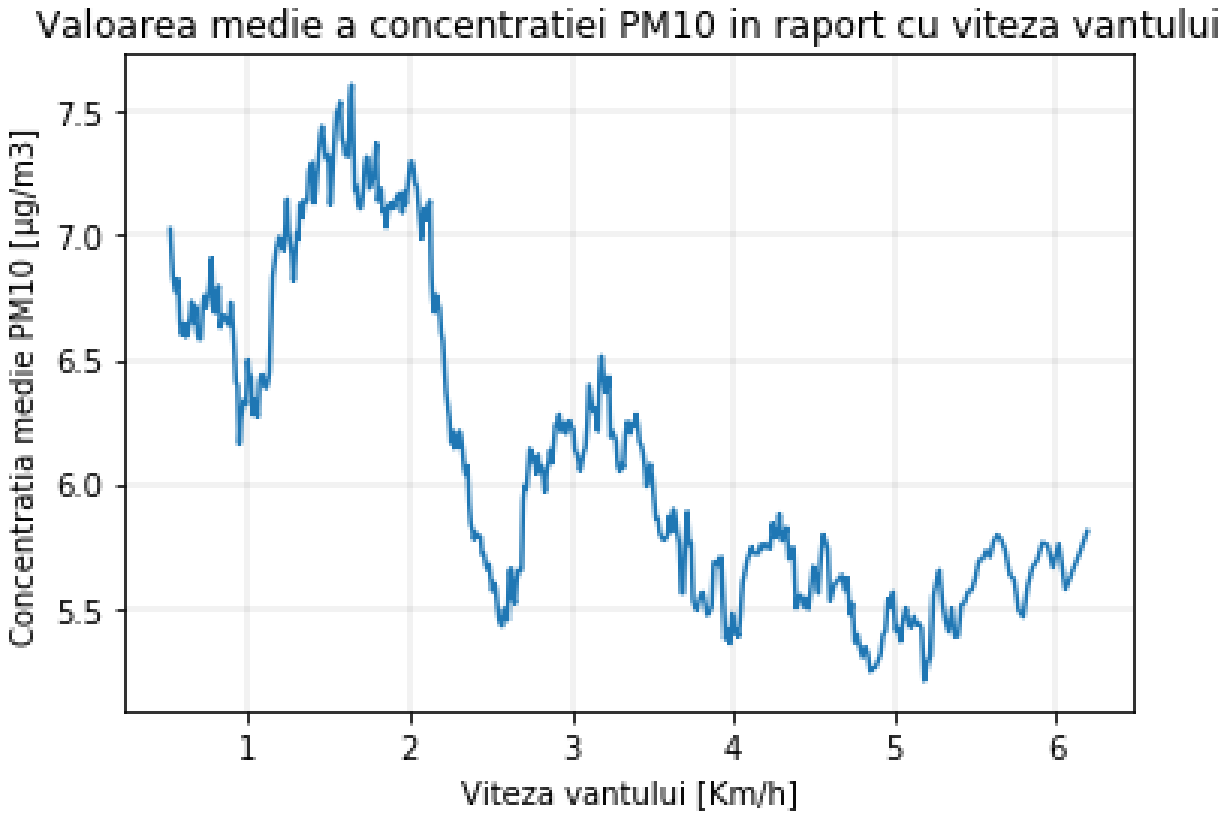
Figură 4.5

In acest grafic se poate observa relația dintre concentrația poluantului PM10 si viteza vântului, pentru București in intervalul 20 Aprilie ora 15:08 si 23 Aprilie ora 16:09. In aceasta perioada viteza vântului a atins un maxim de 8.75 Km/h, iar concentrația de PM10 un maxim de 51.02 µg/m3.

Pentru a ajunge la un model care sa aproximeze cat mai bine modul in care aceste valori se vor schimba in următoarea ora, valorile colectate anterior se vor procesa, aducându-le in formatul necesar. Pentru antrenarea modelului sunt necesare date de intrare si de ieșire. Datele de intrare vor fi viteza vântului si concentrația curenta de poluant, iar cele de ieșire, concentrația de poluant ora următoare.

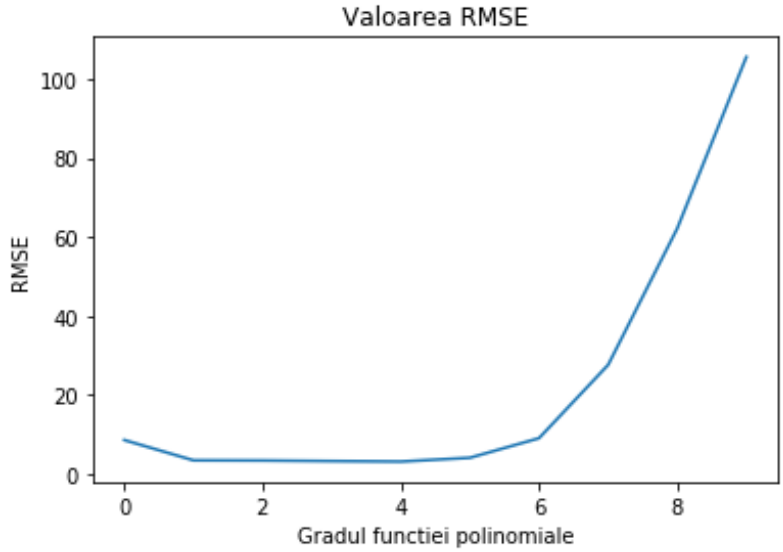
Regresia se va realiza folosind bibliotecile specializate din python, si anume „sklearn” pentru modelul regresiei si „pandas” pentru procesarea datelor.

Un prim pas in analizarea datelor colectate a fost cel de a reprezenta grafic cantitatea medie de poluant asociata unei anumite viteze a vântului. Se poate observa in graficul de mai jos cum cantitatea medie de poluant scade atunci când viteza vântului creste.



Figură 4.6

In încercarea de a aproxima aceasta relație cat mai bine, vom încerca realizarea regresiei polinomiale pe ecuații de grade diferite, începând cu gradul 0 pana la gradul 10, si se va calcula RMSE (Root Mean Square Error), mărime ce ne va ajuta sa determinam care dintre modele este cel mai bun. Pentru fiecare grad, se va calcula eroarea pătratica medie, pentru fiecare dintre cele 8 componente pe care dorim sa le aproximam, făcând o medie a acestor valori. După evaluarea tuturor modelelor putem reprezenta grafic valorile obținute. In cazul de fata cea mai buna valoare a fost obținuta pentru gradul 4.



Figură 4.7

Pentru a putea folosi acest model in java, trebuie sa cunoaștem coeficienții determinați, care sa poată fi integrați într-o funcție de predicție. Coeficienții au fost introduși in cod in felul următor (pentru a nu încarcă pagina este prezentat un singur exemplu, pentru PM10).

```
private static double pm10Coef[] = {0.0,-
0.5723140624957793,0.7578966461767227,0.23357257005112653,0.17097189146586853,0.009688017571
218932,-0.039286179429026694,-0.03050615634049534,-
0.011457938774469404,0.0002895032453965351,0.0019571788782336034,0.0018496234849557083,0.000
9939104352038905,0.0001437894902788637 5,-1.0111364551267561e-05};

private static double pm10Intercept = 0.7099155843068061;
```

Iar funcția de predicție este următoarea:

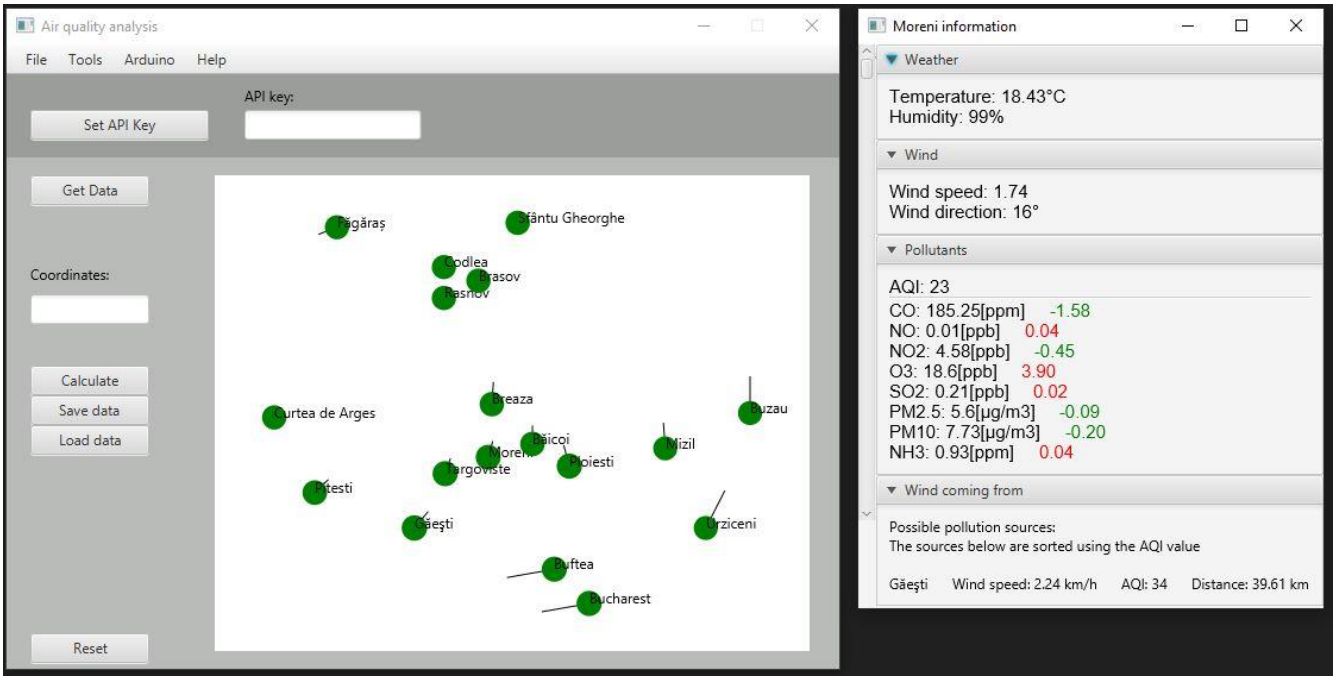
```
public static double predictPm10(double currentWind, double currentPm10)
  {
      double a = currentWind;
      double b = currentPm10;


      double terms[] = Utils.genTerms(a, b, 4);


      double val = 0;
      for(int i = 0; i < 15; i++)
      {
          val += pm10Coef[i]*terms[i];
      }
```

```
        return val + pm10Intercept;
    }
```

După ce toate operațiile de procesare a datelor sunt finalizate, utilizatorul poate selecta o stație dorita, moment in care se va deschide o fereastra noua, cu datele relevante.



Figură 4.8

Fereastra din dreapta oferă informații detaliate despre stație, interfața fiind împărțita in segmente ce pot fi strânse, in cazul in care utilizatorul nu dorește sa încarce spațiul cu acestea. Primele informații sunt Temperatura si umiditatea, unde se poate observa o temperatura nu foarte ridicata, si o umiditate de 99% datorata precipitațiilor din momentul colectării datelor. Următorul segment prezinta condițiile vântului, anume viteza si direcția in grade, 0 fiind Nord iar 180 Sud. Al treilea segment conține cantitățile de poluanți curente si predicția acestora in ora următoare, valorile fiind colorate sugestiv: verde in cazul in care concentrația scade, o concentrație scăzută fiind un lucru bun si roșu in cazul in care aceasta creste. Înainte de valorile concentrațiilor este scris AQI-ul, o valoare de baza in analiza calității aerului. Ultimul segment al interfeței prezinta datele despre posibila sursa de poluare, fiind listate localitățile cele mai apropiate, unde vântul este direcționat spre localitatea curenta. Acestea sunt ordonate descrescător in funcție de valoarea AQI-ului, deoarece daca acesta este mare, șansele ca o cantitate de poluare care poate fi transportata cresc, calitatea aerului in jurul stației actuale putând scădea.

Conectarea la API

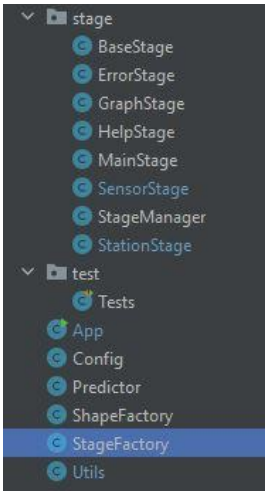
Folosind un serviciu care prezinta funcționalități ce pot necesita efectuarea unei plăti, acesta necesita crearea unui cont. In urma acestei operații utilizatorul va primi o cheie de acces al API-ului. Aplicația realizata stochează in memorie cheia care oferă funcționalitățile de baza într-o clasa speciala numita „Config”. Aceasta clasa conține date despre căile către directoarele folosite la salvarea fișierelor, dimensiuni standard ale interfețelor, si cheia de acces către API.

In cazul in care utilizatorul dorește folosirea unui alt plan fata de cel standard oferit, cheia de acces poate fi schimbata folosind câmpul „API key”, in care o noua cheie poate fi introdusa, iar apăsarea butonului „Set API key” va realizarea salvarea unei noi chei. In cazul in care câmpul este gol, dar se apasă butonul, cheia va reveni la starea inițiala.

Detalii relevante

Una din funcționalitățile utile ale aplicației este închiderea recursiva a ferestrelor deschise. De exemplu, daca utilizatorul deschide mai multe ferestre care prezinta informații despre stații, in momentul închiderii aplicației acestea vor fi închise automat. Fără aceasta operație, programul nu ar termina execuția pana când toate ferestrele nu ar fi închise, lucru care ar crea disconfort in utilizare.

In dezvoltarea aplicației, codul a fost structurat si împărțit in clase si pachete dedicate funcțiilor necesare. Fiecare clasa creata îndeplinind atribuții cat mai clare, iar clasele cu funcții asemănătoare au fost grupate in pachete cu nume sugestive. De exemplu pentru afișarea diferitelor ferestre au fost create mai multe clase, fiecare fiind specializata in afișarea unui singur tip de fereastra. Pentru a crea instanțe a acestor clase a fost folosit design pattern-ul „factory”, folosind clasa „StageFactory” care construiește clase precum „ErrorStage”, „MainStage”, etc.



Figură 4.9

4.3. Senzorii

Prezentare generala

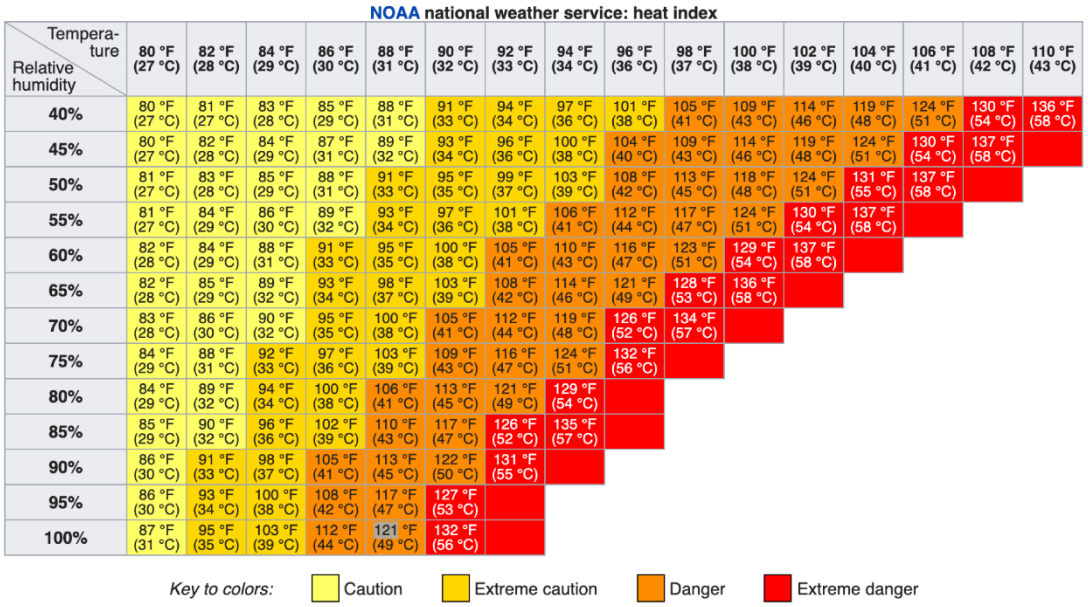
Pentru a putea colecta date din locația exactă a utilizatorului se vor folosi patru senzori conectați la o placa Arduino UNO. Senzori vor putea colecta date despre temperatura, umiditate, presiune si concentrația anumitor gaze care, daca se afla într-o concentrație prea mare in aerul respirat pot scădea calitatea acestuia. Dintre senzorii folosiți, doi dintre aceștia sunt capabili sa masare temperatura ambientala, ceea ce poate fi util, servind ca o măsura de siguranța, astfel daca un senzor nu funcționează corect se va putea observa o diferența destul de mare intre cele doua temperaturi raportate. Umiditatea este o alta mărime relevanta pentru confort, o valoare adecvata aflandu-se in intervalul 40% - 60%.

Mărimile măsurate

Măsurând temperatura si umiditatea relativa putem calcula o noua mărime importanta, si anume indexul de căldura. Aceasta mărime are rolul de a oferi o descriere mai buna a modului in care temperatura este resimțita de o persoana (se considera o valoare a temperaturii măsurate într-o zona umbrita). Deoarece oamenii folosesc ca mecanism de răcire transpirația, o umiditate suficient de mare poate încetini procesul de evaporare a apei (transpirației), ducând astfel la senzația de supraîncălzire. Indexul de căldura poate si resimțit diferit de la persoana la persoana, fiind influențat de diverși factori cum ar fi condiția fizica, nivelul de hidratare, etc. In proiectul prezentat, aceasta mărime este calculata in mod automat folosind o biblioteca specializata pentru senzorul DHT11, apelând funcția prezentata mai jos. Ultimul parametru al funcției poate lua valoarea *true* sau *false*, indicând daca valorile vor fi in grade Fahrenheit sau Celsius.

```
float heatIndex_dht = dht.computeHeatIndex(temperature_dht, humidity_dht, false);
```

Mai jos poate fi observat un tabel cu valorile posibile împărțite in patru categorii de risc, acestea fiind foarte importante deoarece un index ridicat poate avea efecte fatale.



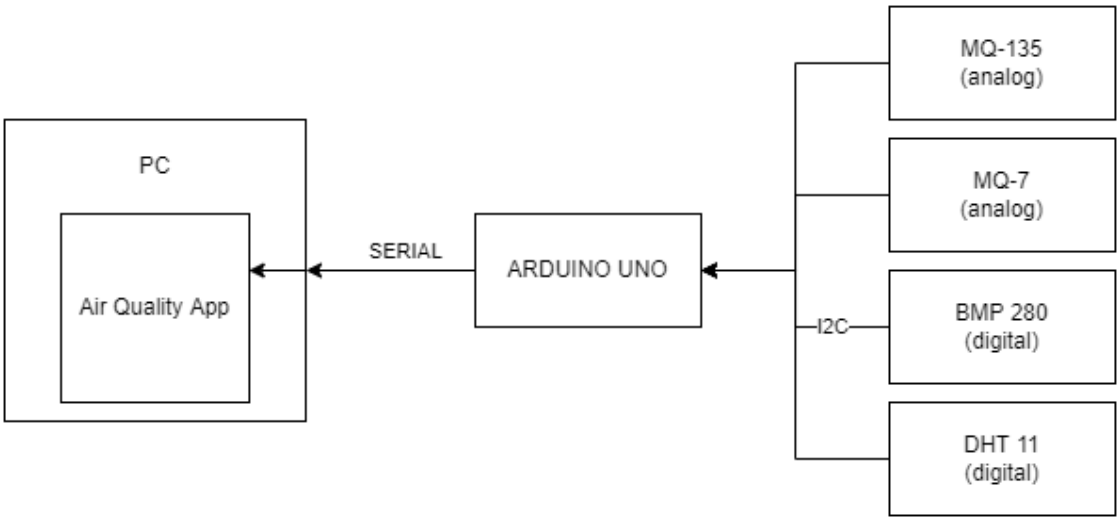
Figură 4.10

Senzorii MQ sunt folosiți pentru a măsura concentrația diverselor gaze din aer, precum monoxidul de carbon sau amoniacul, ambele fiind compuși toxici si foarte periculoși, efecte grave asupra sănătății. Concentrația acestora este direct dependenta de tensiunea măsurata la ieșirea senzorilor, rezistenta interna a acestora fiind influențata de concentrația gazelor din jur. Deoarece senzorii MQ reprezintă o varianta mai puțin costisitoare, măsurătorile realizate nu vor putea fi perfecte, elementele interne fiind sensibile si la alte gaze din atmosfera.

Temperatura este măsurata folosind senzorii DHT11, dar si BMP280, putând astfel determina o valoare a temperaturii reale mai apropiata e cea adevărata. Pe lângă temperatura senzorul DHT11 este folosit pentru a măsura umiditatea relativa din aer, iar BMP280 pentru a măsura presiunea atmosferica.

Structura hardware

Senzorii sunt conectați la arduino folosind atât pinii digitali cat si analogici. Valorile senzorilor MQ-7 si MQ-135 sunt citite folosind pinii analogici A1 respectiv A0. Senzorul BMP280 comunica cu arduino folosind protocolul I2C, fiind astfel conectat la A4(SDA) si A5(SCL), iar senzorul DHT11 folosește un singur pin digital pentru transmiterea datelor, anume pinul D2. Mai jos poate fi observata schema bloc a proiectului. Toate datele sunt colectate folosind placa Arduino si sunt transmise mai departe către aplicația principala, urmând a fi citite folosind comunicarea seriala si portul la care este conectat Arduino.



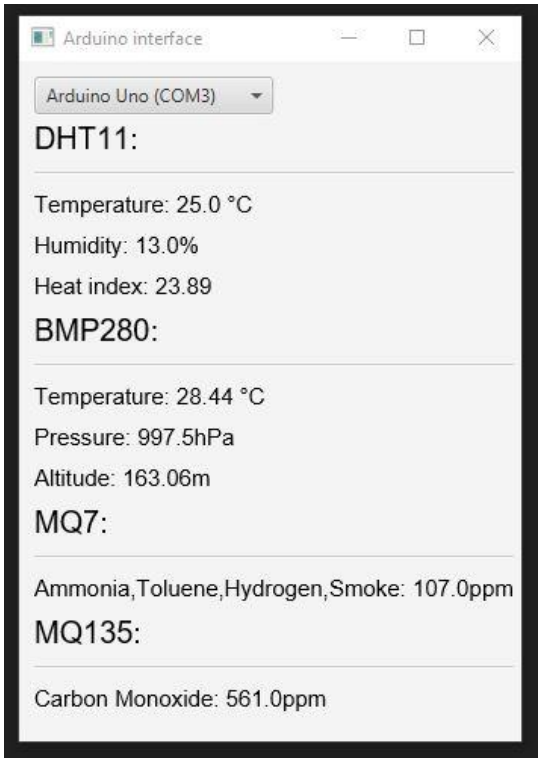
Figură 4.11

Interfața

Aplicația prezinta o interfața pentru a facilita conectarea cu placa si observarea activității senzorilor. Interfața poate fi accesata din meniul „Arduino” al interfeței principale, unde, dintr-un meniu „drop-down” se poate alege dispozitivul cu care sa se stabilească conexiunea, in cazul de fata dispozitivul poate fi doar o placa Arduino. Daca nu este găsit un dispozitiv corespunzător, in meniul de selecție, va exista o singura opțiune cu numele „Nothing avalible on port”, altfel se va afișa numele corespunzător, de exemplu „Arduino Uno (COM3)”.

Interfața este împărțita in patru segmente, folosind linii orizontale, delimitând astfel o zona pentru fiecare senzor. Deasupra fiecărei linii este numele senzorului, iar sub aceasta datele colectate de la respectivul dispozitiv. De exemplu, cum se poate observa si in figura, senzorul DHT11 oferă informații despre temperatura, umiditate, si valoarea indexului de căldura.

O problema vizibila in figura este diferența dintre temperatura măsurata de cei doi senzori, 25.0 °C si 28.44 °C. Aceasta problema poate apărea din cauza amplasării senzorilor, BMP280 fiind poziționat mai aproape de cei doi senzori MQ, care au in componenta lor un element ce produce căldura, ducând astfel la o măsurare nu foarte precisa a temperaturii ambientale. Un alt factor care poate produce aceasta eroare este izolarea mai slaba decât cea a senzorului DHT11. Pentru a remedia acest lucru senzorii MQ trebuie poziționați la o distanta suficient de mare de senzori de temperatura pentru a nu perturba colectarea datelor.



Figură 4.12

Comunicarea seriala

Pentru a putea primi si procesa datele de la Arduino, comunicarea cu acesta se realizează pe un thread separat. Acesta este folosit pentru a nu bloca aplicația principala. La fel ca funcționalitatea de colectare automata a datelor, comunicarea cu modulul fizic, care strânge date locale, se poate realiza in paralel cu activitatea desfășurata de utilizator in aplicația principala.

In primul rând utilizatorul trebuie sa aleagă dispozitivul cu care vrea sa se realizeze conexiunea. Pentru a face asta, este necesara selectarea din lista afișata la deschide ara ferestrei prezentate anterior, a dispozitivului Arduino. Programul va scana porturile si in cazul in care nu exista dispozitive disponibile se va afișa mesajul „Nothing available on port”, altfel se afișează numele dispozitivelor disponibile.

Datele se vor trimite in format JSON, pentru a facilita citirea acestora si memorarea lor într-o clasa specializata.

5. Evaluarea rezultatelor

Pentru a evalua rezultatele predicției in ceea ce privește concentrațiile valorilor poluanților, se vor folosi datele înregistrate de aplicație in decursul a 25 de ore. Pentru strângerea acestor date a fost folosita funcționalitatea de colectare automata, iar rezultatele au fost salvate in format text.

Se vor analiza datele legate de concentrația de PM10 începând cu data de 25 Mai 2022, ora 22:51. Deoarece acest fișier conține si informații nedorite, precum înregistrări ale activității aplicației, este necesara preprocesarea lor, păstrând doar informația esențiala. Se pot observa mai jos doua fragmente din fișierul salvat, înainte respectiv după procesare:

Task GET DATA performed at: Wed May 25 22:51:28 EEST 2022

Setting coordinates...

bbox initialize with default data

Constructing the map...

Try to get stations data from API

Getting weather data...

TOTAL API CALLS: 19

Total TIME ELAPSED: 2223ms

Displaying data...

Clearing pane...

Calculating...

Moreni:25/05/2022,1.32,11.31,11.270602523435908

Breaza:25/05/2022,1.7,8.54,8.644055570432997

Băicoi:25/05/2022,1.51,8.28,8.312751948701058

Targoviste:25/05/2022,0.54,10.98,11.141379229560888

Ploiesti:25/05/2022,0.76,15.19,15.422923318263226

Găeşti:25/05/2022,1.08,11.94,11.89993099145064

Rasnov:25/05/2022,2.94,4.65,5.184915421023601

Buftea:25/05/2022,1.03,16.45,16.550537769071912

Mizil:25/05/2022,2.85,7.81,8.663561204585072

Brasov:25/05/2022,2.6,4.93,5.389502231481544

Pitesti:25/05/2022,1.0,11.27,11.227099458533369

Codlea:25/05/2022,2.34,3.53,3.8671141827694053

Curtea de Arges:25/05/2022,0.48,9.88,10.024058720193114

Bucharest:25/05/2022,1.03,20.52,20.62000784232191

Urziceni:25/05/2022,5.39,5.91,7.677970104908151

Sfântu Gheorghe:25/05/2022,1.68,2.79,3.19803751493941

Buzau:25/05/2022,4.66,6.48,8.244420376580958

Făgăraș:25/05/2022,2.34,5.08,5.46438307275092

Task SAVE DATA performed at: Wed May 25 22:51:58 EEST 2022

Saving data...

Saved file: src/main/resources/saves/map_file_25_05_2022_22_51_58.json

După preprocesare ajungând in formatul CSV (Comma-separated values) :

statie,data,vant,pm10,pm10_next

Moreni,25/05/2022,1.32,11.31,11.270602523435908

Breaza,25/05/2022,1.7,8.54,8.644055570432997

Băicoi,25/05/2022,1.51,8.28,8.312751948701058

Targoviste,25/05/2022,0.54,10.98,11.141379229560888

Ploiesti,25/05/2022,0.76,15.19,15.422923318263226

Găeşti,25/05/2022,1.08,11.94,11.89993099145064

Rasnov,25/05/2022,2.94,4.65,5.184915421023601

Buftea,25/05/2022,1.03,16.45,16.550537769071912

Mizil,25/05/2022,2.85,7.81,8.663561204585072

Brasov,25/05/2022,2.6,4.93,5.389502231481544

Pitesti,25/05/2022,1.0,11.27,11.227099458533369

Codlea,25/05/2022,2.34,3.53,3.8671141827694053

Curtea de Arges,25/05/2022,0.48,9.88,10.024058720193114

Bucharest,25/05/2022,1.03,20.52,20.62000784232191

Urziceni,25/05/2022,5.39,5.91,7.677970104908151

Sfântu Gheorghe,25/05/2022,1.68,2.79,3.19803751493941

Buzau,25/05/2022,4.66,6.48,8.244420376580958

Făgăraș,25/05/2022,2.34,5.08,5.46438307275092

In continuare datele vor fi analizate folosind limbajul de programare Python, iar cu ajutorul bibliotecilor specializate, se va calcula, pentru fiecare stație valoarea RMSE (Root Mean Square Error), intre lista valorilor actuale si lista valorilor prezise, urmând ca la final, sa se calculeze o valoare medie ce tine cont de toate stațiile.

Programul va separa datele in așa fel încât sa fie transformate din formatul CVS in doua liste stocate in memoria programului, una cu valori reale, respectiv cu valori prezise. După efectuarea acestor operații rezultatele sunt afișate in formatul următor:

--- valorile RMSE pentru fiecare statie ---

Statia 0 : RMSE = 1.4918834467972628

Statia 1 : RMSE = 1.194369242147286

Statia 2 : RMSE = 0.9362335369247768

Statia 3 : RMSE = 1.751245492060721

Statia 4 : RMSE = 2.5285645631130578

Statia 5 : RMSE = 1.7452552921451268

Statia 6 : RMSE = 0.7970742159820576

Statia 7 : RMSE = 3.2498931769497283

Statia 8 : RMSE = 1.2793660681088979

Statia 9 : RMSE = 1.1441979366756425

Statia 10 : RMSE = 1.9154230419409046

Statia 11 : RMSE = 1.0976633378290621

Statia 12 : RMSE = 1.2254082002101536

Statia 13 : RMSE = 2.9055056637939707

Statia 14 : RMSE = 1.9746231967981436

Statia 15 : RMSE = 0.9673113950010251

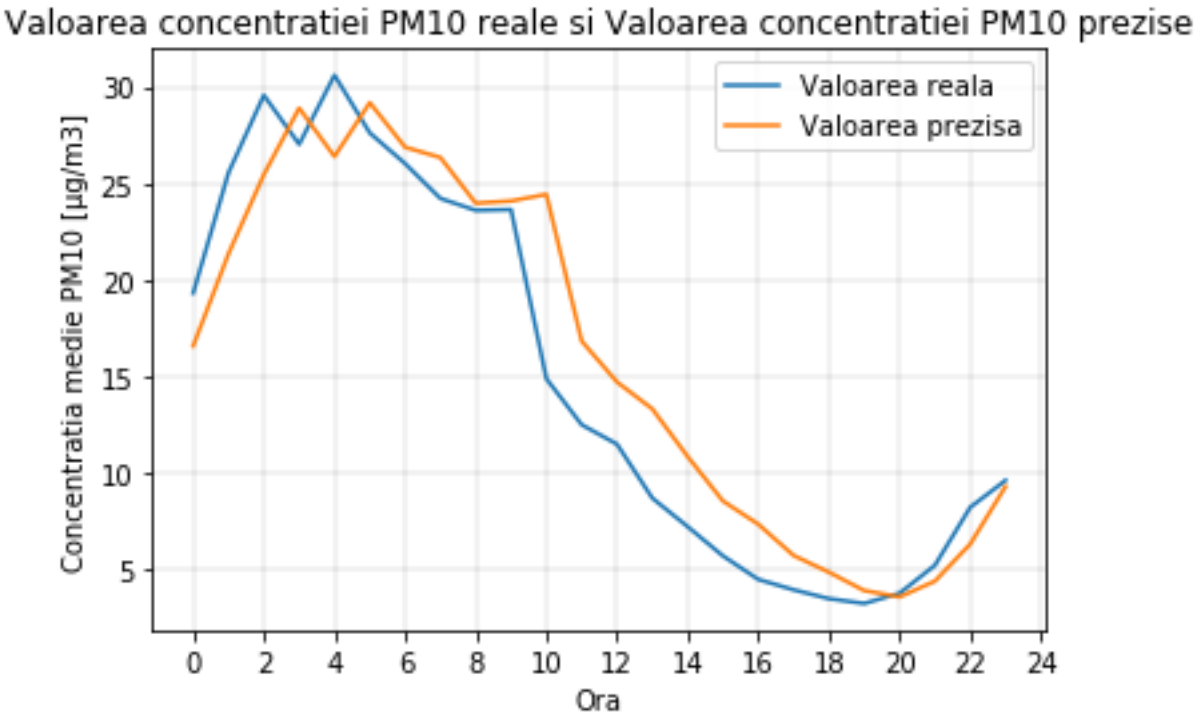
Statia 16 : RMSE = 1.474289481978079

Statia 17 : RMSE = 1.3539396544997628

--- valoarea medie RMSE ---

Mean RMSE: 1.6129026079419808

Pentru a putea vizualiza mai ușor aceste rezultate se poate trasa graficul valorilor, pentru una dintre stații:



Figură 5.1

Ținând cont de imprevizibilitatea condițiilor atmosferice si activității umane, factori care contribuie in mare măsura, in schimbarea cantităților de poluanți din atmosfera, rezultatele obținute sunt apropiate de cele reale, eroarea fiind una relativ mica.

6. Concluzii

In acest capitol se vor prezenta concluziile referitoare capitolelor prezentate mai sus, obiectivele proiectului, rezultatele obținute si posibile îmbunătățiri ce pot fi aduse pe viitor. Așa cum am prezentat de la început, scopul lucrării descrise a fost de a oferi o soluție in ceea ce privește informarea locuitorilor cu privire la calitatea aerului. Aceste informații pot fi utile oricărei persoane, indiferent de grupul de risc din care face parte, expunerea la poluare putând dauna sănătății oricui. Informațiile colectate si prezentate de aplicație sunt temperatura si umiditatea aerului (si indexul de căldura in ceea ce privește modulul fizic), viteza si direcția vântului, cantitate poluanților monitorizați de stațiile meteorologice folosite de platforma OpenWeather, si legăturile dintre orașe, prezentând cele mai apropiate orașe cu un grad de poluare ridicat care poate fi transportat de vânt către o zona de interes.

In ceea ce privește dezvoltarea aplicației putem spune ca scopul a fost atins, aceasta fiind dezvoltata si testata cu succes. Utilizatorului ii este pus la dispoziție un sistem capabil sa colecteze date relevante despre starea vremii si calitatea aerului, in mod manual sau automat, aceasta funcționalitate dovedindu-se utila in momentul antrenării modelului de regresie, si sa le prezinte intr-un mod intuitiv. Aplicația dispune de o interfața simpla ce facilitează interacțiunea cu programul, si oferă o modalitate de a evalua datele in mod vizual, cum am văzut in capitolele anterioare, folosind culori si ferestre separate pentru a prezenta datele.

Modulul fizic al proiectului, ansamblul de senzori conectai la placa de dezvoltare arduino, reprezintă de asemenea o parte importanta a proiectului, aceasta fiind capabila sa colectez date utile, dar care nu pot fi descoperite folosind doar API-ul public pus la dispoziție de platforma OpenWeather si folosit in tot parcursul lucrării. Senzorii oferă o evaluare locala a calității aerului, mai restrânsa si mai simpla decât cea oferita de rețeaua de stații meteorologice, dar importanta. Monitorizarea nivelului de monoxid de carbon (CO) din aerul ce ne înconjoară este extrem de importanta, expunerea chiar si pe termen scurt la o cantitate ce depășește un prag critic (de obicei de peste 200 ppm), putând fi nociva.

Un aspect important este evaluarea rezultatelor obținute, pas descris in detaliu in capitolul anterior. Am putut observa ca aplicația este capabila sa prezică schimbarea cantităților de poluanți considerați in aceasta lucrare (CO, NO, NO2, O3, SO2, PM2.5, PM10, NH3). Data fiind starea imprevizibila a condițiilor meteorologice si a factorilor umani precum traficul sau activitățile industriale, care influențează in mod drastic starea aerului, dar care nu pot fi observate folosind stațiile meteorologice puse la dispoziție, predicția schimbărilor in cantitatea de poluanți putând fi una nu foarte precise. Cu toate acestea rezultatele obținute sunt apropiate de cele reale, oferind astfel, chiar daca nu o valoare precisa, una suficient de buna pentru a putea aproxima calitatea aerului in următoarea ora. Urmărind aceasta schimbare, utilizatorul poate lua decizii in ceea ce privește expunerea la aerul din exterior, decizii care pot duce la evitarea unor probleme de sănătate pe termen lung.

Având in vedere ca datele folosite in antrenarea modelului de regresie au fost colectate folosind aplicația dezvoltata, acest lucru poate reprezenta o problema deoarece volumul de date necesar unei predicții precise nu este suficient de mare. Acest aspect poate fi remediat pe viitor prin colectarea datelor

pe o perioada mai lunga (luni sau ani, luând astfel in considerare modificările aduse de schimbarea de sezon). O alta problema ar putea fi, așa cum am văzut si in capitolul 4, amplasarea senzorilor pe plăcută de test. Căldura radiata de senzorii MQ putând sa afecteze senzorii de temperatura.

7. Bibliografie

[1] Calitatea aerului in Romania, http://www.mmediu.ro/categorie/calitatea-aerului/56

[2] UN, Revision of world urbanization prospects, https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html

[3] Gradul de urbanizare in Romania, http://starea-natiunii.ro/index.php/ro/noutati/facts-figures/25-facts-figures/134-romania-se-mentine-pe-ultimul-loc-in-regiune-in-ceea-ce-priveste-gradul-de-urbanizare

[4] What is Java, https://java.com/en/download/help/whatis_java.html

[5] JDK 11 Documentation, https://docs.oracle.com/en/java/javase/11/

[6] What is JavaFX, https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[7] What is Python, https://www.python.org/doc/essays/blurb/

[8] Scikit guide, https://scikit-learn.org/stable/getting_started.html

[9] Arduino UNO overview, https://docs.arduino.cc/hardware/uno-rev3

[10] Open Electronics, Presenting MQ sensors, https://www.open-electronics.org/presenting-mq-sensors-low-cost-gas-and-pollution-detectors/

[11] DHT11 Datasheet, https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[12] BMP280 Datasheet, https://www.bosch-sensortec.com/media/boschsensortec/downloads/product_flyer/bst-bmp280-fl000.pdf

[13] David Mintz (September 2018). Technical Assistance Document for the Reporting of Daily Air Quality – the Air Quality Index (AQI) (PDF). North Carolina: US EPA Office of Air Quality Planning and Standards. EPA-454/B-18-007. Retrieved 15 September 2021

https://www.airnow.gov/sites/default/files/2020-05/aqi-technical-assistance-document-sept2018.pdf

8. Lista contribuțiilor personale

Aplicație pentru analiza calității aerului Student: Aldea Andrei Coordonator: Prof. Radu Hobincu		
	Activitate	Durata [zile]
1	Documentare tehnologii necesare in realizarea proiectului	3
2	Realizarea unui prototip al aplicației pentru testarea funcționalităților de baza	5
3	Dezvoltarea codului proiectului	10
4	Realizarea modulului hardware	3
5	Testarea si verificarea datelor colectate de senzori	1
6	Dezvoltarea codului necesare predicției poluanților	2
7	Antrenarea si evaluarea modelului de predicție	4
8	Finalizarea proiectului, modificări finale	2
9	Redactarea lucrării	6
	Total	36

Durata este exprimată în zile echivalente de lucru, adică 1 zi = 8 ore

9. Anexe

Anexa 1 – Codul folosit pentru colectarea datelor cu Arduino.

```
#include <Adafruit_BMP280.h>
#include "DHT.h"

#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
Adafruit_BMP280 bmp; // I2C Interface

int coA = A1;
int qA = A0;

void setup() {
  Serial.begin(9600);
  Serial.println(F("![TEST] DHT11"));
  Serial.println(F("![TEST] BMP280"));

  dht.begin();
  boolean bmpStatus = bmp.begin(0x76);
  Serial.println("![STATUS]      BMP280="      +
String(bmpStatus));

  if (!bmpStatus) {
    Serial.println(F("![ERROR] Could not find a
valid BMP280 sensor, check wiring!"));
    while (1);
  }

  /* Default settings from datasheet. */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,
/* Operating Mode. */
                  Adafruit_BMP280::SAMPLING_X2,
/* Temp. oversampling */
                  Adafruit_BMP280::SAMPLING_X16,
/* Pressure oversampling */
                  Adafruit_BMP280::FILTER_X16,
/* Filtering. */
Adafruit_BMP280::STANDBY_MS_500);   /*   Standby
time. */

  pinMode(coA, INPUT);
  pinMode(qA, INPUT);
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  //read data from BMP
  float temperature_bmp = bmp.readTemperature();
  float pressure_bmp = bmp.readPressure()/100;
  float  altitude_bmp  =  bmp.readAltitude(1017);
//The pressure(hPa) at sea level

  //read data from DHT
  float temperature_dht = dht.readTemperature();
  float humidity_dht = dht.readHumidity();

  //read data from MQ-135
  float co2comp = analogRead(coA);
  float co2ppm = map(co2comp,0,1023,400,5000);

  //read data from MQ-7
  float q = analogRead(qA);

  // Check if any reads failed and exit early (to
try again)
  if(isnan(temperature_dht)                    ||
isnan(humidity_dht))
  {
    Serial.println(F("Failed  to  read  from  DHT
sensor!"));
    return;
```

```
  }


  if(isnan(temperature_bmp)                    ||
isnan(pressure_bmp) || isnan(altitude_bmp))

  {

    Serial.println(F("Failed  to  read  from  BMP
sensor!"));

    return;

  }


  // Compute heat index in Celsius (isFahreheit
= false)

  float             heatIndex_dht             =
dht.computeHeatIndex(temperature_dht,
humidity_dht, false);

  Serial.println(

    "{\"temperature_dht\":          "          +
String(temperature_dht) + ", " +

    "\"humidity_dht\": " + String(humidity_dht)
+ ", " +

    "\"heatIndex_dht\":            "           +
String(heatIndex_dht) + ", " +

    "\"temperature_bmp\":           "          +
String(temperature_bmp) + ", " +

    "\"pressure_bmp\": " + String(pressure_bmp)
+ ", " +

    "\"altitude_bmp\": " + String(altitude_bmp)
+ ", " +

    "\"co2ppm\": " + String(co2ppm) + ", " +

    "\"q\": " + String(q) + "}"

  );
```

Anexa 2 – Codul folosit in analizarea rezultatelor

```
#%% init

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.metrics import mean_squared_error


# Importing the dataset

datas     =     pd.read_csv('data_predict_2.csv',
encoding= 'unicode_escape')

w, h = 25, 18

data = [[0 for x in range(w)] for y in range(h)]


#%%

curr_station = 0

max_station = h


curr_pos = 0


for index, row in datas.iterrows():


    data[curr_station][curr_pos] = [row['pm10'],
row['pm10_next']]

    curr_station += 1


    if curr_station == max_station:

        curr_station = 0;

        curr_pos += 1



rmse_list = []


print("--- valorile RMSE pentru fiecare statie -
--")


for i in range(h):

    station_data = data[:][i]


    y = []

    y_pred = []


    for j in range(w):

        y.append(station_data[j][0])

        y_pred.append(station_data[j][1])


    y.pop(0)
```

```
    y_pred.pop(-1)

    rmse     =     np.sqrt(mean_squared_error(y,
y_pred))

    rmse_list.append(rmse)

    print("Statia", i, ": RMSE =", rmse)


print("")

print("--- valoarea medie RMSE ---")

print("Mean RMSE:", np.mean(rmse_list))


#%%

station_data = data[:][7]


y = []

y_pred = []

for j in range(w):

    y.append(station_data[j][0])

    y_pred.append(station_data[j][1])


y.pop(0)

y_pred.pop(-1)


plt.title("Valoarea concentratiei PM10 reale si
Valoarea concentratiei PM10 prezise")

plt.xlabel("Ora")

plt.ylabel("Concentratia medie PM10 [µg/m3]")

plt.grid(color='black',           linestyle='-',
linewidth=0.1)

plt.plot(y)

plt.plot(y_pred)

plt.xticks(np.arange(0, len(y)+1, 2))

plt.legend(["Valoarea     reala",     "Valoarea
prezisa"])

plt.show()
```

Anexa 3 - Codul folosit in căutarea coeficienților de regresie

```
#%% init

import numpy as np

import pandas as pd

from         sklearn.linear_model         import
LinearRegression

from         sklearn.preprocessing        import
PolynomialFeatures

from        sklearn.model_selection       import
train_test_split

from sklearn.metrics import mean_squared_error




DEG = 4


#date,h,name,wind_speed,temp,hum,co,no,no2,o3,so
2,pm2_5,pm10,nh3,co   next,no   next,no2   next,o3
next,so2 next,pm2_5 next,pm10 next,nh3 next


# Importing the dataset

datas  =  pd.read_csv('csv\data_.csv',  encoding=
'unicode_escape')


min_simple = 0


#%% SIMPLE TEST

print("\nCAUTARE COEFICIENTI")


#input data

co_in    = datas.iloc[:, [3, 6]] .values

no_in    = datas.iloc[:, [3, 7]] .values

no2_in   = datas.iloc[:, [3, 8]] .values

o3_in    = datas.iloc[:, [3, 9]] .values

so2_in   = datas.iloc[:, [3, 10]].values

pm2_5_in = datas.iloc[:, [3, 11]].values

pm10_in  = datas.iloc[:, [3, 12]].values

nh3_in   = datas.iloc[:, [3, 13]].values
```

```
#out data

co_out    = datas.iloc[:,14].values

no_out    = datas.iloc[:,15].values

no2_out   = datas.iloc[:,16].values

o3_out    = datas.iloc[:,17].values

so2_out   = datas.iloc[:,18].values

pm2_5_out = datas.iloc[:,19].values

pm10_out  = datas.iloc[:,20].values

nh3_out   = datas.iloc[:,21].values



#regres

poly      =      PolynomialFeatures(degree=DEG,
include_bias=True)

poly_features = poly.fit_transform(co_in)

X_train,    X_test,    y_train,    y_test    =
train_test_split(poly_features,          co_out,
test_size=0.3, random_state=42)



poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)

print(poly.get_feature_names())



poly_reg_y_predicted                              =
poly_reg_model.predict(X_test)



poly_reg_rmse                                     =
np.sqrt(mean_squared_error(y_test,
poly_reg_y_predicted))

print("co", poly_reg_rmse)

coef    =    ",".join([str(i)    for    i    in
poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")









poly      =      PolynomialFeatures(degree=DEG,
include_bias=True)

poly_features = poly.fit_transform(no_in)

X_train,    X_test,    y_train,    y_test    =
train_test_split(poly_features,          no_out,
test_size=0.3, random_state=42)



poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)



poly_reg_y_predicted                              =
poly_reg_model.predict(X_test)



poly_reg_rmse                                     =
np.sqrt(mean_squared_error(y_test,
poly_reg_y_predicted))

print("no", poly_reg_rmse)

coef    =    ",".join([str(i)    for    i    in
poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")









poly      =      PolynomialFeatures(degree=DEG,
include_bias=True)

poly_features = poly.fit_transform(no2_in)

X_train,    X_test,    y_train,    y_test    =
train_test_split(poly_features,          no2_out,
test_size=0.3, random_state=42)



poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)



poly_reg_y_predicted                              =
poly_reg_model.predict(X_test)



poly_reg_rmse                                     =
np.sqrt(mean_squared_error(y_test,
poly_reg_y_predicted))

print("no2", poly_reg_rmse)

coef    =    ",".join([str(i)    for    i    in
poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")









poly      =      PolynomialFeatures(degree=DEG,
include_bias=True)
```

```
poly_features = poly.fit_transform(o3_in)

X_train,    X_test,    y_train,    y_test    =
train_test_split(poly_features,          o3_out,
test_size=0.3, random_state=42)



poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)



poly_reg_y_predicted                              =
poly_reg_model.predict(X_test)



poly_reg_rmse                                     =
np.sqrt(mean_squared_error(y_test,
poly_reg_y_predicted))

print("o3", poly_reg_rmse)

coef    =    ",".join([str(i)    for    i    in
poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")
```

```
poly      =      PolynomialFeatures(degree=DEG,
include_bias=True)

poly_features = poly.fit_transform(so2_in)

X_train,    X_test,    y_train,    y_test    =
train_test_split(poly_features,         so2_out,
test_size=0.3, random_state=42)



poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)



poly_reg_y_predicted                              =
poly_reg_model.predict(X_test)



poly_reg_rmse                                     =
np.sqrt(mean_squared_error(y_test,
poly_reg_y_predicted))

print("so2", poly_reg_rmse)

coef    =    ",".join([str(i)    for    i    in
poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")
```

```
poly      =      PolynomialFeatures(degree=DEG,
include_bias=True)

poly_features = poly.fit_transform(pm2_5_in)

X_train,    X_test,    y_train,    y_test    =
train_test_split(poly_features,       pm2_5_out,
test_size=0.3, random_state=42)



poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)



poly_reg_y_predicted                              =
poly_reg_model.predict(X_test)



poly_reg_rmse                                     =
np.sqrt(mean_squared_error(y_test,
poly_reg_y_predicted))

print("pm2_5", poly_reg_rmse)

coef    =    ",".join([str(i)    for    i    in
poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")
```

```
poly      =      PolynomialFeatures(degree=DEG,
include_bias=True)

poly_features = poly.fit_transform(pm10_in)

X_train,    X_test,    y_train,    y_test    =
train_test_split(poly_features,        pm10_out,
test_size=0.3, random_state=42)



poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)



poly_reg_y_predicted                              =
poly_reg_model.predict(X_test)



poly_reg_rmse                                     =
np.sqrt(mean_squared_error(y_test,
poly_reg_y_predicted))

print("pm10", poly_reg_rmse)
```

```
coef = ",".join([str(i) for i in poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")
```

```
poly = PolynomialFeatures(degree=DEG, include_bias=True)

poly_features = poly.fit_transform(nh3_in)

X_train, X_test, y_train, y_test = train_test_split(poly_features, nh3_out, test_size=0.3, random_state=42)
```

```
poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train, y_train)


poly_reg_y_predicted = poly_reg_model.predict(X_test)


poly_reg_rmse = np.sqrt(mean_squared_error(y_test, poly_reg_y_predicted))

print("nh3", poly_reg_rmse)

coef = ",".join([str(i) for i in poly_reg_model.coef_])

print(coef)

print("interc:", poly_reg_model.intercept_)

print("")
```