Universitatea Națională de Știință și Tehnologie POLITEHNICA București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Stație Meteo bazată pe IoT

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de
Inginer în domeniul ”Electronică, Telecomunicații si Tehnologia Informației”
Programul de studii de licență ”Calculatoare și Tehnologia Informației”

Conducător științific

Prof. Dr. Ing. Radu Rădescu

Absolvent

Tudor-Bogdan Geană

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Program de studiu **INF**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **GEANĂ B.D. Tudor-Bogdan, 442A**

1. Titlul temei: Stație meteo bazată pe IoT

2. Descrierea temei şi a contribuţiei personale a studentului (în afara părţii de documentare):
Tema își propune crearea modelului unei stații meteo care să stocheze informațiile acumulate într-o bază de date, aceasta fiind însoțită de o aplicație web pentru afișarea și interacționarea cu datele. Dispozitivul de măsurare va consta dintr-un microcontroler cu capacități Bluetooth conectat la o serie de senzori pentru a colecta datele atmosferice. Senzorii folosiți: de umiditate, temperatură, presiune atmosferică, intensitate luminoasă, prezența ceții și măsurare a vitezei vântului. Aceștia vor transmite datele către calculatorul care joacă rolul de host al bazei de date, folosind conexiune serială. Mai departe, prin intermediul unui script Python, datele transmise pe portul serial vor fi introduce automat în baza de date. Sistemul de gestiune al bazelor de date folosit este MySQL. Stocarea datelor se va realiza la bordul dispozitivului, folosind un RaspberryPi drept calculator host. La nivel local, prin intermediul Virtual Network Computing, utilizatorul poate vizualiza baza de date. Tot la nivel local, utilizatorul poate cere afișarea datelor sub forma unor grafice și afișarea diferitelor valori statistice importante unei baze de date, (tipul distribuției, media, dispersia). Partea de backend a aplicației va fi realizată folosind framework-ul Django al limbajului Python. Pentru partea de frontend a aplicației se vor folosi tehnologiile HTML, CSS, JavaScript (și librăria D3.js). Aceste date vor putea fi încărcate într-un model de machine learning pentru a obține o predicție pentru următoarea zi, acest rezultat fiind disponibil local și pe pagina web.

3. Discipline necesare pt. proiect:
Microcontolere, Circuite Integrate Digitale, Arhitectura Sistemelor de Calcul, Programare Orientată pe Obiecte, Intefețe Om-Mașină

4. Data înregistrării temei: 2024-03-08 05:45:46

Conducător(i) lucrare,
Prof. dr. ing. Radu RĂDESCU

Student,
GEANĂ B.D. Tudor-Bogdan

Director departament,
Ş.L. dr. ing Bogdan FLOREA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **421ae10595**

Anexa 6

Anexa 5

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Stație Meteo bazată pe IoT*", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *domeniul* "Electronică, Telecomunicații si Tehnologia Informației", programul de studii *program "Calculatoare și Tehnologia Informației"* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referințăla sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilorîn vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe carele prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, *27.06.2024*

Absolvent *Tudor-Bogdan Geană*

_____

(semnătura în original)

Cuprins

Listă de tabele

Listă de figuri

Listă de acronime

I2C	Inter-Integrated Circuit
ADC	Analog to Digital Converter(Convertor Numeric Analogic)
API	Application Programming Interface
ARM	Advanced RISC Machine
CPU	Central Processing Unit(Unitatea centrală de procesare)
CSS	Cascading Style Sheets
GND	Ground
GPU	Grapics Processing Unit(Unitatea de procesare grafică)
HAT	Hardware Attached on Top(componente atașate deasupra)
hPa	HectoPascal
HTML	Hyper Text Markup Language
JSON	JavaScript Object Notation
LED	Light Emitting Diode(Diodă ce emite lumină
LSTM	Long Short Term Memory
MAE	Mean Absolute Error(Eroarea pătratică absolută)
MSE	Mean Squared Error(Eroare pătratică medie)
RMSE	Root Mean Squared Error
RPM	Rotații pe minut
SCL	Serial Clock Line(Fir serial pentru ceas)
SDA	Serial Data Line(Fir serial pentru date)
SGDB	Sistem de gestionare a bazelor de date
TOPS	Terra Operații pe secundă
URL	Uniform Resource Locator
XML	eXtensible Markup Language

Introducere

Analiza condițiilor meteorologice a devenit în ultima perioadă de timp un subiect foarte important pe plan global datorită schimbărilor atmosferice produse de rapida evoluție a societății în care trăim, mai ales din punct de vedere industrial. Este important ca oamenii să poată conștientiza aceste schimbări, fie că ele sunt pozitive sau negative, pentru a putea face dezicii informate cu privire la modul în care interacționează cu mediul înconjurător. Consider că, pentru a putea atinge un asemenea scop, este foarte important că oricine să poată avea acces la un dispozitiv ușor de utilizat, care le va permite utilizatorilor să observe și studieze schimbările mediul înconjurător pe cont propriu, având acces la seturi de date lipsite de bias.

Un dispozitiv bazat pe IoT(Internet of Things) reprezintă o formă de tehnologie care are senzori, programe și o conexiune la Internet, având rolul de a colecta și transmite date în rețea. Costul acestui tip de dispozitive este unul redus, ceea ce le face disponibile pentru utilizatorii domestici.

Aceste două concepte menționante anterior se pot combina foarte ușor pentru a obține o stație meteo bazată pe IoT. Astfel, datele acumulate de această stație meteo pot fi făcute disponibile într-o rețea prin intermediul unei interfețe web, permițând un anume grad de libertate în amplasarea dispozitivului. De asemena, prețul redus al componentelor permite utilizarea mai multor senzori de același fel pentru a asigura redundanța sistemului. De asemenea, consumul redus de putere al acestor componente crește și mai mult ușurința de utilizare a unui astfel de ansamblu.

Pentru stocarea eficientă și de durata a datelor, un astfel de dispozitiv va folosi o bază de date controlată de un sistem de gestionare a bazelor de date. Astfel, se va asigura faptul că toate datele sunt stocate corespunzător, iar valorile meteorologice sunt asociate momentelor de timp de măsurare potrivite. Aceste date pot astfel fi folosite foarte ușor atât pentru calcule, cât și pentru a ilustra tendințe prin intermediul graficelor și al tabelelor.

Predicțiile meteorologice clasice se fac prin urmărirea mișcării maselor de aer plus alte fenomene meteorologice cheie. Acest proces necesită aparatură specializată care este extrem de costisitoare, greu de folosit, inacesibilă și cu rezultate greu de interpretat. O altă soluție pentru predicții meteorologice care și-a făcut apariția în ultimii ani este utilizarea rețelelor neurale recurente care au abilitatea de a face predicții bazate pe orodonarea temporală a datelor. Mai concret, un asemenea model poate observa tendințe pe axa temporală care ar putea fi invizibile unei persoane.

Noile dezvoltări hardware permit inferențe locală a unui model antrenat anterior folosind niște dispozitive ASIC, cunoscute sub numele de Acceleratoare de AI. Aceste dispozitive au capacitatea de a efectua un număr extrem de ridicat de operații matematice(13 TOPS în cazul celui folosit în acest proiect), permițând calculatoarelor să infereze un model eficient fără a consuma din resursele procesorului, care ar fi folosite cu un grad de eficiență scăzut pentru această sarcină.

Fiind un dispozitiv IoT, așa cum a fost menționat și anterior, această stație meteo va disponibiliza datele măsurătorilor către utilizator prin intermediul unei interfețe web. Pentru această parte, vor fi folosite librării care facilitează vizualizarea datelor în moduri cât mai sugestive. În combinație cu această interfață se află un API care respectă constrângerile arhitecturale REST, asigurând astfel acecsul la date într-o formă cât mai facil de utilizat pentru partea de frontend(formatul JSON).

Toate aceste aspecte menționate anterior sunt implicate în realizarea proiectului meu, mai exact o **Stație Meteo Bazată pe IoT**, iar dezvoltarea acestuia poate fi împărțită în 4 etape principale.

Etapa 1: Elementul de senzoristică. Stația meteorologică are abilitatea de a înregistra date despre umiditate, presiune, temperatură, viteza vântului, intensitatea luminoasă și ceață. Acestea sunt conectate la 2 microcontrolere care au rolul de a multiplexa semnalul senzorilor și ale converti într-un format care poate fi foarte ușor parsat de către unitatea centrală. Senzorii sunt găzduiți în incinte specializate, care minimizează efectele negative ale mediului și asigură măsurători cât mai precise.

Etapa 2: Construirea bazei de date. Folosind sistemul de gestionare al bazelor de date MariaDB, se va construi o bază de date care poate stoca ieșirile tuturor senzorilor, atât in formatul *crud*(raw data) cât și în cel procesat. Mai departe, aceste date devin accesibile către orice utilizator care are acces.

Etapa 3: Antrenarea modelului de predicție. Folosind date meteorologice achiziționate între anii 2018 și 2024, se va putea antrena un model de învățare automată ce are abilitatea de a prezice condițiile meteo ce urmează. Predicțiile se vor face local, astfel încât eficiența operației crește enorm. De asemenea, rezultatele prezise pot fi comparate cu rezultatele obținute de la senzori pentru a aprecia eficiența modelului. Rețeaua neurală recurentă folosită este LTSM, care permite observarea datelor și tendințelor din acestea în relație cu trecerea timpului.

Etapa 4: Construirea interfeței Web. Fiind un dispozitiv bazat pe IoT, este imperativ ca acesta să aibă o interfață web care să poată permite vizualizarea datelor și interacțiunea cu sistemul. Pentru partea de frontend se va folosi combinația de HTML, CSS și JavaScript pentru a obține o interfață responsivă și plăcută vizual. De asemenea, folosind librăria D3.js, vizualizarea datelor va deveni un lucru facil, având la dispoziție numeroase variante de grafice. Partea de backend va fi construită folosind framework-ul Django al limbajului Python, obținând astfel un API care respectă constrângerile arhitecturale REST.

1.Prezentarea Resurselor Hardware

1.1 Descrierea secțiunii de lucrare

În această secțiune a lucrării, voi prezenta senzorii care sunt folosiți în construcția stației meteo pentru a acumula datele din mediul înconjurator. Dispozitivul este echipat cu senzori pentru măsurarea presiunii, umidității, temperaturii, lumini, vitezei vântului precum și un senzor de detecție a prezenței ceții.

Pentru fiecare senzor va fi realizată o descriere generală, urmată de o prezentare a construcției și funcționării senzorului, iar la final va fi elaborat modul în care senzorul comunică cu microcontroller-ul.

1.2 Senzor de presiune atmosferică– > BMP180

1.2.1 Descrierea generală și trăsături importante ale senzorului

BMP180 este un senzor creat de compania Bosch și folosit pentru a măsura presiunea și temperatura din mediul înconjurător. Acesta poate fii conectat la un microcontroller prin intermediul interfeței I^2C. Este un senzor care functionează la putere mica și tensiune scăzută, fiind perfect pentru aplicații de consum mic. Poate măsura presiuni cu valorile aflate între 300 și 1100 de **hPa**. Tensiunea de alimentare(V_{DD}) care poate varia între 1.8 și 3.6 **V**, iar tensiunea de referință pentru I^2C (V_{DDIO}) se află în plaja de valori 1.62 – 3.6 **V**. Valoarea consumului standard furnizată de producator este de 5 **µA** pentru o rată de măsură de un eșantion pe secundă.

Presiunea măsurabilă	300 - 1100 hPa
Tensiunea de alimentare VDD	1.8 - 3.6 V
Tensiunea de alimentare VDDIO	1.62 - 3.6 V
Consumul	5 µA / eșantion / secundă

Tabel 1.1 - Caracteristici BMP180 [15]

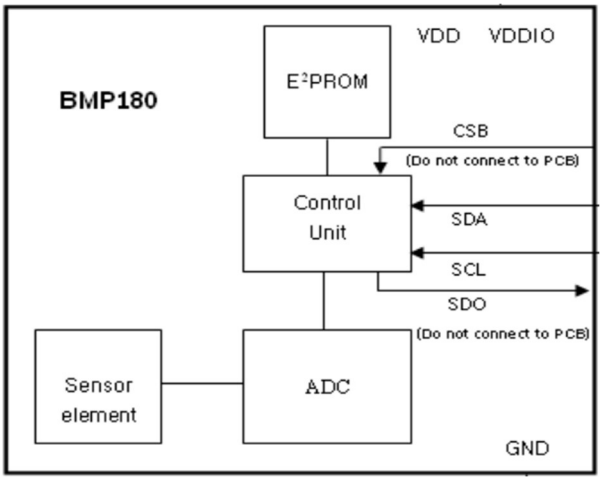
1.2.2 Numerotarea PIN-ilor și ce reprezintă fiecare

Pin	Function
1	VCC
2	GND
3	SCL
4	SDA

Tabel 1.2 - PINOUT BMP180 [15]

1.2.3 Construcția senzorului

Senzorul este alcătuit dintr-un piezo-rezistor, un convertor analog-digital și o unitate de control dotată cu interfața I^2C. EEPROM-ul este folosit pentru a stoca datele utilizate la calibrare. Piezo-rezistorul are atașată o membrană pentru a crea "elementul de senzor" ce se poate vedea în figură. Acest element transformă variațiile din presiunea atmosferică în semnale electrice, care sunt mai departe trimise către ADC pentru eșantionare. Aceste valori sunt modificate pe baza datelor de calibrare și trimise de către unitatea de control.



Figură 1.1 - Construcția BMP180 [15]

1.2.4 Funcționarea senzorului

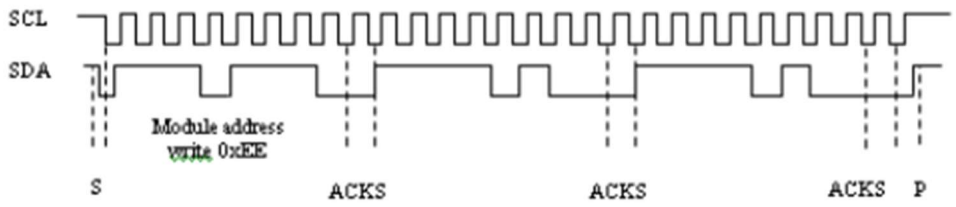
Microcontroller-ul trimite o secventa de start pentru a începe o măsurătoare de presiune. Datele sunt achiziționate, iar, după un timp de conversie, valoarea obținută poate fi citită prin intermediul interfeței I^2C. În documentație valoarea este numită UP și valoarea poate fi scrisă ca un număr pe 19 biți. Timpul de conversie este de aproximativ 4.5 ms. Pentru a putea obține un output în hPa se vor folosi datele de calibrare.

Figură 1.2 – Funcșionarea BMP180 [15]

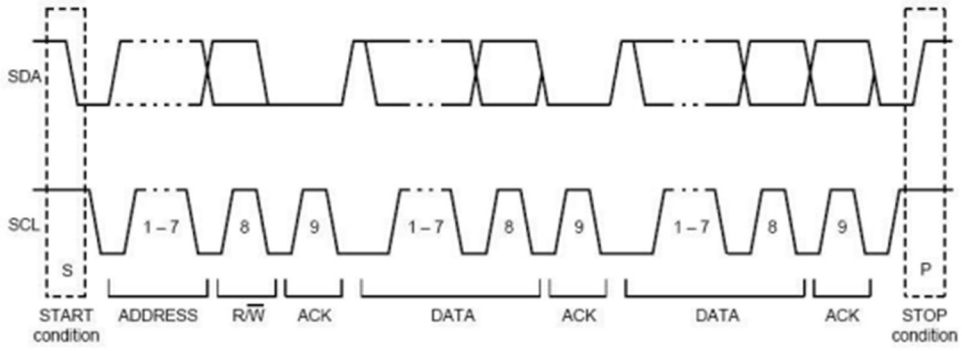
1.2.5 Interfața și protocolul de transmisie

Este o interfață digitală ce lucrează pe 2 fire. Aceasta este folosită pentru a controla senzorul, a accesa datele de calibrare și a citi datele de ieșire ale senzorului. Frecvența maxima de lucru a ceasului este 3.4Mbit/sec, având disponibil atât modul rapid cât și cel de mare viteză. Atât SCL cât și SDA sunt ieșiri cu drenă deschisă.

Protocolul acestei interfețe are anumite trăsături speciale care asigură sincronizarea. Există condițiile de start și de stop, care sunt definite astfel: condiția de start este atunci când SDA este pe front descrescător și SCL are valoarea logică **HIGH**, iar condiția de stop este atunci când SDA este pe front crescător, iar SCL are valoarea logică **HIGH**. Datele pot fi adăugate doar atunci când SCL are valoarea logică **LOW**, pentru a evita hazardele combinaționale.



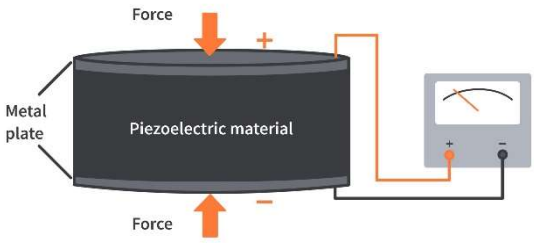
Figură 1.4 - Schema de sincronizare [15]



Figură 1.3 - Interfața I2C [15]

1.2.6 Fenomenul fizic

Fenomenul fizic care stă la baza funcționării acestui senzor este efectul piezoelectric, mai exact capacitatea anumitor materiale să capete un potential electric în urma aplicării unei forte mecanice și viceversa. Prima atașarea unei membrane la un piezo cristal aceasta devine sensibilă la schimbările din presiunea atmosferică. Este o tehnologie asemănătoare cu ce se găsește și în microfoane. Acest efect este cauzat de separarea spontană a sarcinilor în cadrul unui material.



Figură 1.5 - Efectul piezoelectric [1]

1.3 Anemometru -> Motor DC

1.3.1 Descrierea generală și trăsături importante ale motorului

În general, un motor DC este folosit pentru a transforma energia electrică în energie mecanică, cu scopul de a roti ansamblul atașat motorului. În contextul acestei lucrări, motorul DC este folosit pe post de anemometru, pentru a putea măsura viteza vântului. Acesta va avea atașate niște cupe menite să capteze vântul, care va învârti motorul. Acest concept este cunoscut sub numele de tachometru, mai exact un apart care produce o tensiune pe bazată pe numărul de rotații al dispozitivului.

Viteza la 3V	20000 RPM
Curentul la 3V	350 mA
Curentul de oprire la 3V	4 A
Puterea la oprire la 3V	0.00529 N*m

Tabel 1.3 - Specificații anemometru

1.3.2 Numerotarea PIN-ilor și ce reprezintă fiecare

Pin	Function
Roșu	Semnal
Negru	GND

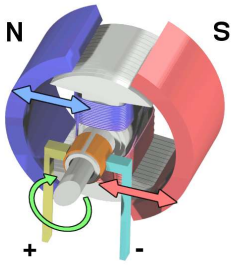
Tabel 1.4 - PINOUT anemometru

1.3.3 Fenomenul Fizic [1]

În momentul în care asupra rotorului unui motor DC aplicăm o energie mecanică, forța electromotoare induce o tensiune în bobinele motorului, tensiune ce este cunoscută sub numele de "back EMF". Tensiunea de ieșire a motorului se calculează pe baza formulei:

$$U_i = K_E \times \omega$$

unde ω este viteza de rotire a motorului și K_E este constanta asociată motorului, care poate fi extrasă din foaia de catalog(în cazul motorului folosit valoarea constantei este de 0.00132). Din această formulă se poate extrage viteza, măsurată în rotații pe minut.



Figură 1.6 - Reprezentarea unui motor DC

1.3.4 Calculul vitezei vântului

Pentru a putea calcula viteza vântului, va trebui întâi să luăm în considerare dimensiunile ansamblului de cupe ce face parte din anemometru. Raza cercului pe care il descrie o cupă a anemometrului este de 10 cm, ceea ce înseamnă ca putem calcula lungimea cercului descris de aceste cupe folosind formula de calcul:

$$L = 2\pi \mathrm{R}$$

Pe baza acestui calcul se va obține o lungime a cercului egală cu 62.832 cm. Astfel, putem afirma că la fiecare rotație, o cupă parcurge distanța calculată anterior. Această valoare este mai departe înmulțită cu numărul de rotații pe minut pentru a putea obține o valoare estimate a vitezei vântului.

1.3.5 Interfața

PIN-ul de *semnal* al acestui motor va fi conectat într-unul din PINii microcontroler-ului care este echipat cu un convertor analog-digital. Semnalul analogic care iese din motor va fi transformat în măsurători individuale are acestui parametru, care vor fi adăugate ulterior în baza de date.

1.4 Senzor de umiditate și temperatură -> DHT22

1.4.1 Descrierea generală și trăsături importante ale senzorului

DHT22 este un senzor digital accesibil, utilizat frecvent pentru măsurarea temperaturii și umidității. Acesta include un senzor capacitiv pentru umiditate și un termistor pentru măsurarea temperaturii aerului înconjurător, oferind un semnal digital pe pinul de date. În comparație cu modelul anterior DHT11, DHT22 oferă o precizie mai mare și un interval de măsurare extins. Poate măsura umiditatea de la 0% la 100% cu o precizie de ±2-5% și temperatura de la -40°C la +80°C cu o precizie de ±0.5°C. Datorită fiabilității, ușurinței în utilizare și protocolului de comunicație simplu, DHT22 este foarte popular în aplicații precum stații meteo, sisteme de monitorizare a mediului și proiecte de automatizare a locuinței.

Tensiunea de alimentare	3.3 - 3.6 V
Limitele măsurabile	umiditate 0 ~ 100%, temperatură -40 ~ 80 C
Rezoluția măsurătorilor	umiditate 0.1 %, temperatură 0.1 C
Durata măsurătorii	în medie 2 secunde

Tabel 1.5 - Specificații DHT22 [17]

1.4.2 Numerotarea PIN-ilor și ce reprezintă fiecare

Pin	Function
1	VDD----power supply
2	DATA--signal
3	NULL
4	GND

Tabel 1.6 - PINOUT DHT22 [17]

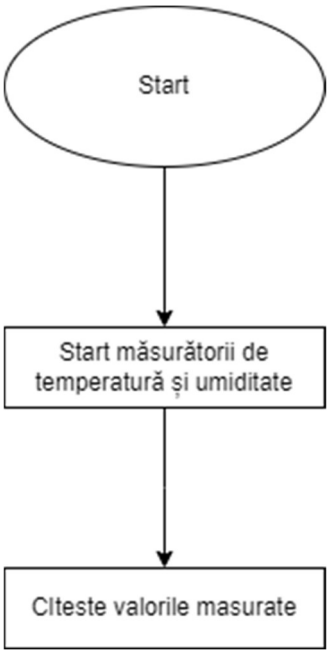
1.4.3 Construcția senzorului

La interior, este alcătuit dintr-un element capacitiv, un termistor, un circuit de conversie și o interfață digital. Elementul capacitiv are rolul de a măsura umiditatea. Ca orice condensator, acesta este alcătuit din două armături și un material dielectric la mijloc. Acest

ansamblu are abilitatea de a își modifica capacitatea pe baza umidității relative a aerului din jur. Termistorul este o rezistență a cărui valoare se modifică în funcție de temperatură, putând astfel să modeleze comportamentul temperaturii. Semnalele brute de la ambele elemente de senzoristică vor fi procesate și transformate în semnale digitale, care pot fi trimise către microcontroller.

1.4.4 Funcționarea senzorului

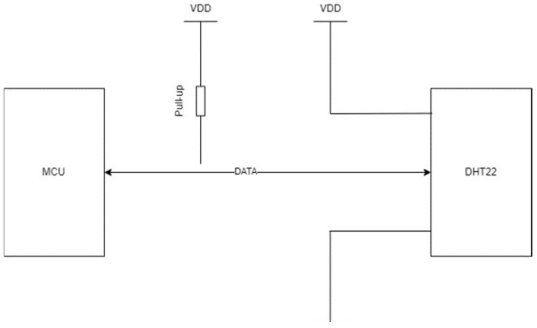
După primirea unui semnal de start, senzorul trece în starea de funcționare(standby status -> running status). După ce semnalul de start a fost primit, senzorul va răspunde cu un semnal de 40 de biți care reprezintă valorile de umiditate și temperatură măsurate. Datele măsurate vor fi trimise doar ca răspun la un semnal de start de la microcontroller. După așteptarea unui interval de 2 secunde, care reprezintă durata unei măsurători, procesul se repetă



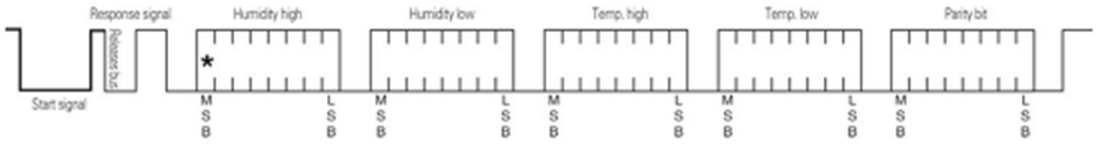
Figură 1.7 - Functionarea DHT22

1.4.5 Interfața

Comunicația dintre microcontroller și senzor se face printr-un singur fir de date. Acesta necesită un rezistor de pull-up pentru a funcționa corect, iar comunicația este una serială.



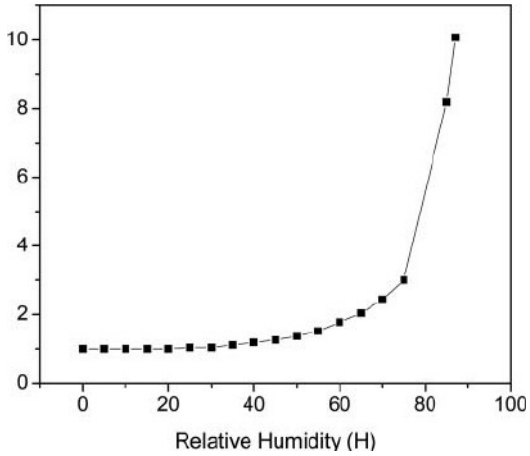
Figură 1.8 - Structura DHT22



Figură 1.9 - Semnalul de la DHT22

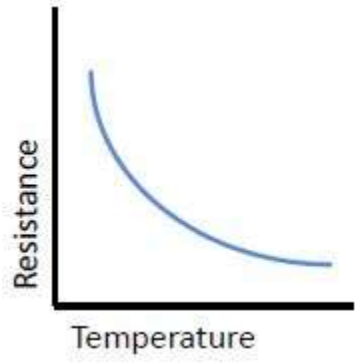
1.4.6 Fenomenul Fizic

Capacitatea unui condensator este puternic legată de umiditatea din mediul înconjurător, în principal prin absorbția în dielectric care duce în schimbări de conductivitate. Aceste schimbări de capacitate pot fi urmărite prin intermediul senzorului și transformată în valori de umiditate.



Figură 1.10 - Influența umidității asupra capacității

Termistorul este un rezistor care își modifică valoarea rezistenței datorită variațiilor din temperatura ambientală. Acesta este construit din materiale speciale care îi dau această proprietate. La fel ca și la condensatorul anterior, termistorul are o caracteristică care ne permite să apreciem valoarea temperaturii ambientale bazat pe rezistența acestuia.



Figură 1.11 - Caracteristica unui termistor NTC

1.5 Senzor de intensitate luminoasă

1.5.1 Descrierea generală și trăsături importante ale senzorului

Senzorul folosit pentru măsurarea intensității luminoase se bazează pe un fotorezistor și pe efectul pe care îl are acesta în funcție de intensitatea luminoasă la care este expus. Acesta oferă flexibilitate în utilizare și permite măsurarea intensității luminoase în multe situații.

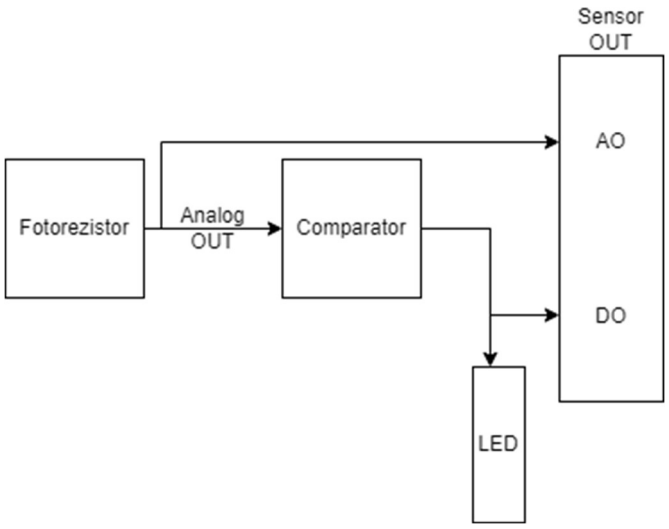
1.5.2 Numerotarea PIN-ilor și ce reprezintă fiecare

Pin	Function
1	VCC
2	GND
3	DO
4	AO

Tabel 1.7 - PINOUT senzor de lumină

1.5.3 Funcționarea senzorului

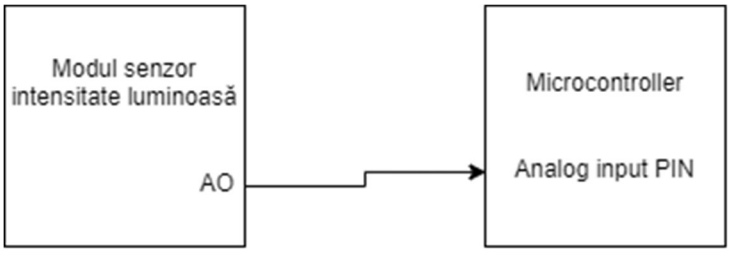
Fotorezistorul este o componenta electronică a cărei rezistență se modifică prin intensitatea luminii care pica asupra lui. Ieșirea acestui fotorezistor este trecută printr-un comparator pentru a obține un output care să fie procesat de microcontroller. Modulul folosit în acest proiect are două LED-uri, unul pentru a indica alimentarea si altul pentru a indica ieșirea comparatorului(mai exact cand ieșirea este mai mare decât pragul).



Figură 1.12 - Construcția senzorului de lumină

1.5.4 Interfața

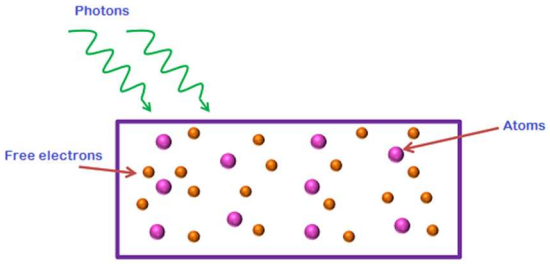
Modulul interacționează cu senzorul prin intermediul a doi PINi: AO și DO. Deoarece în proiectul acesta va fi nevoie de o valoarea concretă a temperaturii în orice moment de timp, va fi folosit PIN-ul AO, care oferă un semnal analogic ce poate fi convertit în temperatură.



Figură 1.13 – Interfața senzorului de lumina cu microcontrolerul

1.5.5 Fenomenul Fizic

Anumite materiale semiconductoare au capacitatea de a își schimba conductibilitatea în momentul în care sunt expuse la lumină. În momentul în care este expus la lumină, anumiți electroni de valență absorb energia de la lumină, eliberând electroni. Acești electroni cară sarcină electrică, modificând astfel conductivitatea materialului.



Figură 1.14 – Electronii liberi din material

1.6 Senzor pentru detecția ceții– > APDS-9930 [2]

1.6.1 Descrierea generală și trăsături importante ale senzorului

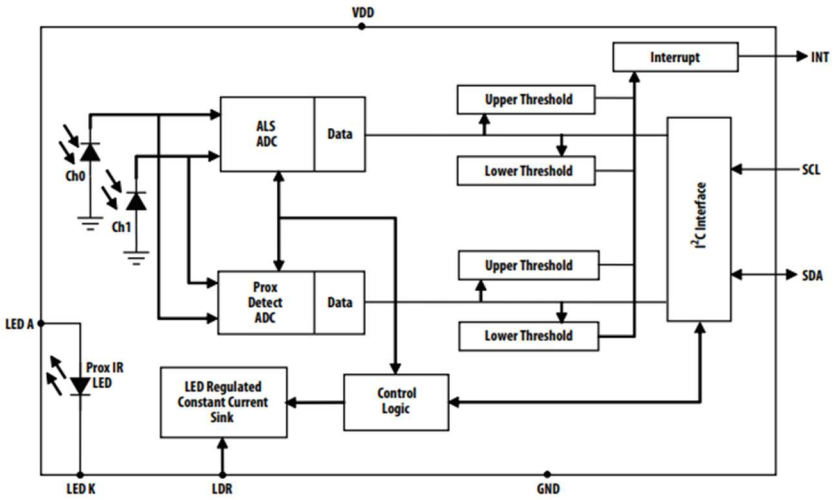
APDS-9930 este un senzor avansat care integrează un senzor de lumină ambientală (ALS) și un senzor de proximitate într-un singur dispozitiv compact. Acest senzor este utilizat pe scară largă în smartphone-uri, tablete și alte electronice de consum pentru a îmbunătăți interacțiunile cu utilizatorul și gestionarea consumului de energie. ALS-ul său poate măsura o gamă largă de intensități ale luminii, de la 0,01 lux la 10.000 lux, cu o sensibilitate ridicată și control automat al câștigului. Senzorul de proximitate detectează obiecte la distanțe de la câțiva milimetri până la 100 mm folosind un LED IR integrat și filtre optice.

1.6.2 Numerotarea PIN-ilor și ce reprezintă fiecare

Pin	Function
1	VL
2	GND
3	VCC
4	SCL
5	SDA
6	INT

Tabel 1.8 - PINOUT APDS-9930

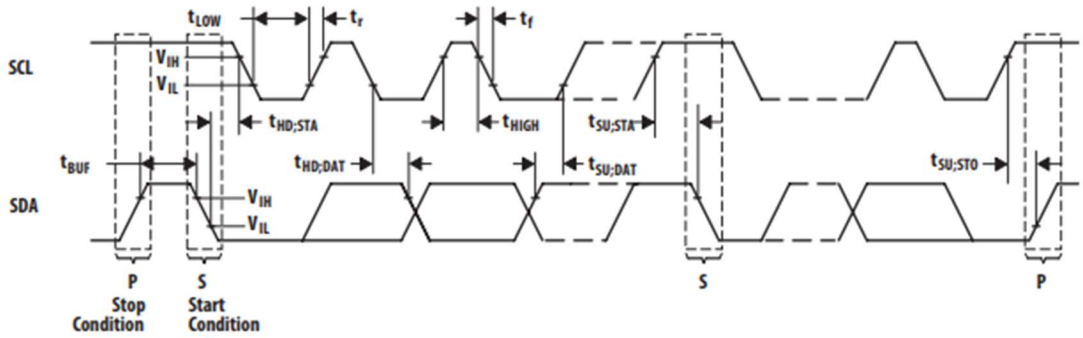
1.6.3 Construcția senzorului



Figură 1.15 - Schema bloc APDS-9930

1.6.4 Interfața și protocolul de transmisie

Protocolul acestei interfețe are anumite trăsături speciale care asigură sincronizarea. Există condițiile de start și de stop, care sunt definite astfel: condiția de start este atunci când SDA este pe front descrescător și SCL are valoarea logică **HIGH**, iar condiția de stop este atunci când SDA este pe front crescător, iar SCL are valoarea logică **HIGH**. Datele pot fi adăugate doar atunci când SCL are valoarea logică **LOW**, pentru a evita hazardele combinaționale.



Figură 1.16 - Diagrma de sincronizare I2C

1.6.5 Detecția de ceață

Funcția principală a acestui senzor este de a detecta proximitatea față de alte obiecte, folosind lumina infraroșie. Una dintre slăbiciunile acestei medtode de măsurare a distanței este că anumite medii pot reflecta undele infraroșii, rezultând astfel rezultate eronate. Ceața reprezintă unul dintre aceste medii, făcând senzorul să detecteze un obiect aflat foarte aproape de acesta, chiar dacă nimic nu este acolo. Acest procedeu, combinate cu o analiză a intensității de lumina, permite sistemului să înregistreze apariția ceții.

1.7 Raspberry Pi 5

1.7.1 Descrierea generală și trăsături cheie

Echipat cu un procesor quad-core Arm Cortex-A76 de 64 de biți, care rulează la 2,4GHz, Raspberry Pi 5 oferă o creștere a performanței CPU de 2-3 ori față de Raspberry Pi 4. În plus, experimentează o îmbunătățire substanțială a performanței grafice datorită GPU-ului VideoCore VII care rulează la 800MHz; are suport pentru ieșirea dual de display 4Kp60 prin HDMI; și suport de ultimă generație pentru camere, datorită unui procesor de semnal de imagine Raspberry Pi restructurat. Aceste caracteristici asigură o experiență fluidă pe desktop pentru consumatori și deschid calea către noi aplicații pentru clienții industriali.

Pentru prima dată, acesta este un computer Raspberry Pi de dimensiuni complete care utilizează siliciu produs intern la Raspberry Pi. Componenta "southbridge" RP1 asigură majoritatea capacităților de I/O pentru Raspberry Pi 5 și oferă o schimbare semnificativă în performanța și funcționalitatea perifericelor. Lățimea de bandă USB agregată este mai mult decât dublă, oferind viteze de transfer mai rapide pentru unitățile externe UAS și alte periferice de mare viteză; interfețele dedicate MIPI pentru camere și display-uri din modelele anterioare, cu două canale de 1Gbps, au fost înlocuite cu un pereche de transceivere MIPI cu patru canale de 1.5Gbps, triplând banda totală și suportând orice combinație de până la două camere sau display-uri; performanța maximă a cardurilor SD este dublată prin suport pentru modul de viteză înaltă SDR104; și pentru prima dată, platforma expune o interfață PCI Express 2.0 cu un singur canal, oferind suport pentru periferice de bandă largă.

Figură 1.17 - RaspberryPi

1.7.2 Interfața USB și conexiunea la microcontroler

La momentul actual, USB este cea mai folosită interfață în conectarea perifericelor la un calculator, beneficiind de funcționalități de tipul *plug and play*, abilitatea de a le conecta fără a reseta calculatorul și asigurând și alimentarea dispozitivelor conectate. RaspberryPi este dotat cu 2 USB-uri de tip 3.0, ce pot atinge viteze de până la 5Gbits/secunda(standard cunoscut sub numele de *Super Speed*), care vor fi folosite pentru a realiza o conexiune serială între calculator și microcontrolere. Odată cu conectarea la un dispozitiv USB, unitatea de lucru crează un port serial(serial port adapter) care acționează ca un buffer pentru informația primită pe această cale.

Procesor	Broadcom BCM2712 2.4GHz quad-core 64-bit Arm Cortex-A76 CPU
Unitate de procesare grafică	VideoCore VII GPU
Output Video	2 x miniHDMI 4Kp60
Wi-Fi	Dual-band 802.11ac Wi-Fi
Bluetooth	Bluetooth 5.0 / Bluetooth Low Energy(BLE)
Conexiuni USB	2 x USB 3.0, suportă operații paralele de 5Gbps / 2 x USB 2.0
Alimentarea	5V/5A curent continuu prin USB-C
I/O	Header cu 40 de pini

Figură 1.18 - Caracteristici RaspberryPi 5 [20]

1.8 M.2 Pi HAT

HAT-urile(Hardware Attached on Top) sunt componente electronice create pentru a fi folosite împreună cu un RaspberryPi, oferind calculatorului funcționalități noi. Pentru lucrearea mea voi folosi un HAT care îmi permite să atașez dispozitive ce folosesc interfața M.2, mai exact un accelerator de AI produs de compania Hailo. Acest HAT se folosește de connectorul PCIe de pe RaspberryPi și îl transformă în M.2, putând fi folosit atât dispozitivele 2230 cât și cele 2242, care consumă maxim 3A de curent. [3]

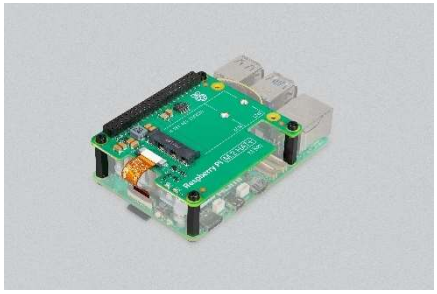


Figure 1.1 - M.2 HAT

1.9 Hailo - 8L AI Accelerator

Un accelerator de AI este o componentă hardware specializată care are rolul de a crește viteza de execuție a proceselor ce implică rețele neurale și algoritmi de învățare automată. Acesta este un ASIC, mai exact un circuit integrat construit și optimizat cu un singur scop în minte(în acest caz optimizarea operațiilor asociate învățării automate cum ar fi înmulțiri de matrice și convoluții). În cadrul proiectului, această componentă hardware va fi folosită pentru a realiza inferența în timp real și local, asigurând astfel faptul că acest dispozitiv nu va trebui să stea mereu conectat la un server pentru a putea realiza predicția de vreme. [4]

Trăsăturile cheie ale acestuia sunt: 13 Tera-Operații pe Secundă(TOPS), temperatură de funcționare între -40 și 85 de grade Celsius, arhitectură ARM și compatibilitatea cu framework-urile cele mai importante de AI cum ar fi TensorFlow si PyTorch. [4]

Figură 1.19 - Hailo - 8L

2 Prezentarea tehnologiilor software

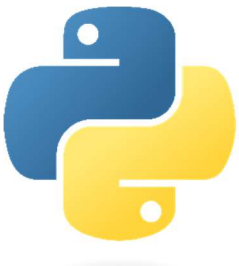
2.1 Descrierea secțiunii de lucrare

Această secțiune de lucrare este dedicată prezentării aspectelor teoretice care însoțesc uneltele software folosite în această lucrare, precum și modul în care acestea sunt folosite și motivația din spatele alegerii acestor tehnologii. Componenta software a acestei lucrări este reprezentată de baza de date locală, pagina web și programarea componentelor electronice.

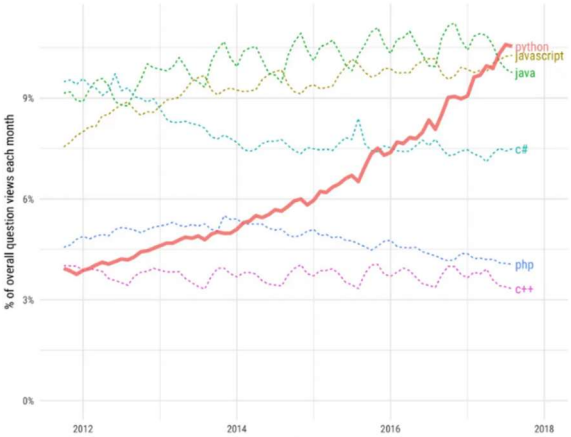
2.2 Python

Python este un limbaj de programare de nivel înalt, orientat pe obiecte și interpretat, cu semantici dinamice. Structurile sale de date avansate, alături de tipizarea și legarea dinamică, îl fac extrem de atractiv pentru Dezvoltarea Rapidă a Aplicațiilor, precum și ca limbaj de scriptare. Sintaxa simplă și ușor de învățat a limbajului Python pune accent pe lizibilitate, reducând astfel costurile de întreținere a programului. Python suportă module și pachete, favorizând modularitatea programului și reutilizarea codului. Interpretorul Python și biblioteca standard extinsă sunt disponibile gratuit în formă sursă sau binară pentru toate platformele principale și pot fi distribuite liber. [5]

Programatorii adesea aleg de Python datorită productivității crescute pe care o oferă. Fiindcă nu necesită un pas de compilare, ciclul de testare-depanare este incredibil de rapid. Depanarea programelor Python este simplă: o eroare sau o intrare greșită nu va cauza niciodată o eroare de segmentare. În schimb, când interpretorul descoperă o eroare, ridică o excepție. Dacă programul nu prinde(catch) excepția, interpretorul afișează eroarea în acel moment fară a afecta codul anterior. întrerupere, parcurgerea codului linie cu linie, și așa mai departe [5].



Figură 2.2 - Logo Python [5]

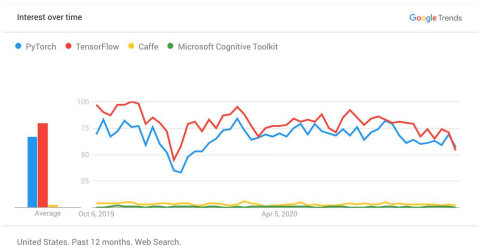


Figură 2.1 - Grafic uzul Python de-a lungul anilor [23]

2.3 PyTorch

PyTorch este o bibliotecă de învățare automată open-source dezvoltată de laboratorul de cercetare AI al Facebook. Este folosită pe scară largă în aplicații precum procesarea limbajului natural și viziunea computerizată. PyTorch se remarcă prin flexibilitatea și ușurința de utilizare, fiind deosebit de apreciat pentru prototiparea rapidă a modelelor de învățare profundă. Oferă două caracteristici principale: calculul tensorilor (similar cu NumPy) cu accelerare puternică prin unități de procesare grafică (GPU), și diferențiere automată pentru construirea și antrenarea rețelelor neuronale. Abordarea sa dinamică a graficului de calcul permite dezvoltarea intuitivă a modelelor, facilitând modificări ale arhitecturii rețelei în timp real. Aceste calități fac din PyTorch o alegere populară printre cercetători și dezvoltatori în scopuri academice și industriale.

Figură 2.3 - Logo Pytorch



Figură 2.4 - Popularitate Pytorch [24]

2.4 HTML5

HTML5 este cea de-a cincea versiune majoră a limbajului HTML, utilizat în mod obișnuit pentru dezvoltarea site-urilor web și a aplicațiilor. Introdus în octombrie 2014, HTML5 aduce o serie de noi funcționalități care facilitează integrarea conținutului multimedia direct în paginile web, eliminând necesitatea plugin-urilor externe, cum ar fi Flash. Este ideal pentru crearea site-urilor web interactive și moderne. Printre inovațiile principale se numără elemente care îmbunătățesc structura și accesibilitatea site-urilor.

De asemenea, HTML5 introduce o varietate de controale pentru formulare și elemente pentru redarea audio și video, precum și API-uri avansate care permit dezvoltarea de aplicații web cu funcții offline și gestionarea eficientă a graficii și multimedia. conținut multimedia, compatibile cu diverse dispozitive și platforme. [6]

Figură 2.5 - Logo HTML5

Figură 2.6 - Beneficii HTML5

2.5 CSS

CSS (Cascading Style Sheets) este un limbaj de stilizare folosit pentru a defini prezentarea vizuală a documentelor HTML. Separând conținutul de aspect, CSS permite dezvoltatorilor web să păstreze un cod HTML mai organizat și mai curat, ceea ce îmbunătățește accesibilitatea și facilitează gestionarea site-ului. Regulile CSS dictează modul în care elementele ar trebui să fie afișate pe diferite tipuri de medii, facilitând crearea de pagini web atrăgătoare vizual. Stilurile sunt implementate folosind selectori care identifică elementele HTML după atribute, cum ar fi clasa, ID-ul sau tipul de element, și proprietăți care definesc valori precum culoarea, tipul de font și structura.

CSS este caracterizat prin caracteristici cum ar fi ordinea în cascadă(de unde și denumirea), care stabilește cum sunt prioritizate și aplicate stilurile în caz de conflict. Limbajul suportă moștenirea, prin care stilurile sunt transmise de la elementele părinte la cele copil, simplificând astfel procesul de stilizare și reducând redundanța de cod. Modelul de CSS(*box model*) este un alt concept de bază, care definește cum sunt gestionate dimensiunile și spațiile în jurul elementelor HTML. Fiecare element este considerat o cutie(*box*), iar înțelegerea proprietăților cum ar fi padding-ul, bordurile și marginile este esențială pentru un control precis al aranjamentului. [7]

Figură 2.7 - Logo CSS

```css
.button {
  background-color: #FF6347;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
  }

.button {border-radius: 12px;}
.button {
  transition-duration: 0.4s;
}

.button:hover {
  background-color: #4CAF50; /* Green */
  color: white;
}
```

Figură 2.8 - Exemplu modelul cutie CSS

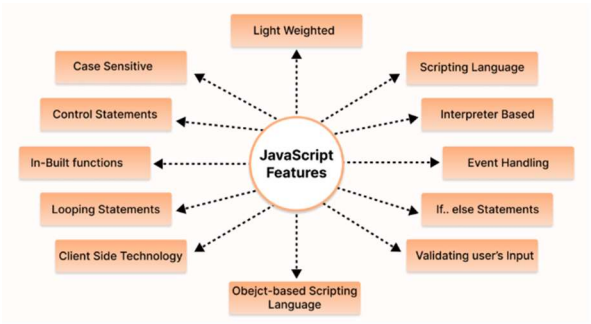
2.6 JavaScript

JavaScript este un limbaj de programare fundamental folosit pe scară largă în dezvoltarea web pentru a crea experiențe interactive pentru utilizatori. Inițial proiectat pentru interacțiuni de partea clientului pe paginile web, JavaScript permite actualizări de conținut în timp real, funcții interactive și animații complexe direct în browserul web.

Evoluția JavaScript l-a văzut adoptând capabilități de partea serverului, în special prin utilizarea Node.js, permițându-i să gestioneze funcționalități backend alături de frontend, făcându-l esențial pentru dezvoltarea aplicațiilor web moderne.

Cu compatibilitatea sa cu programarea orientată pe obiecte, imperativă și declarativă, JavaScript este accesibil pentru începători, dar suficient de puternic pentru dezvoltatorii experți. Este o parte integrantă a stivei de tehnologie web împreună cu HTML și CSS, iar utilitatea sa se extinde la dezvoltarea de aplicații mobile și de desktop, precum și la aplicații pentru dispozitive IoT. [8]

Figură 2.10 - Logo Javascript



Figură 2.9 - Trăsături cheie JavaScript

2.7 MariaDB

MariaDB este un sistem de gestionare a bazelor de date relaționale (SGDB) open-source, recunoscut pentru performanța, fiabilitatea și flexibilitatea sa. Lansat în 2009, a devenit o alegere populară în industria tehnologică pentru dezvoltatori și companii care caută o soluție robustă de bază de date. MariaDB suportă o varietate largă de limbaje de programare și funcționează pe multiple sisteme de operare.

Dispune de un ecosistem bogat de motoare de stocare, plugin-uri și unelte care îi îmbunătățesc funcționalitatea. MariaDB este proiectat să gestioneze eficient seturi mari de date, fiind potrivit pentru totul, de la proiecte mici de dezvoltare până la aplicații de întreprindere de mare amploare.

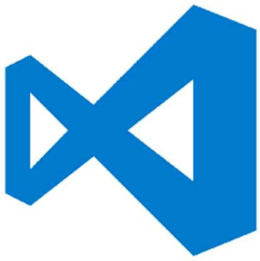
MariaDB este proiectat să fie compatibil cu MySQL, ceea ce înseamnă că suportă aceleași scheme și comenzi ca MySQL. Cu toate acestea, MariaDB include caracteristici și extensii noi care nu se găsesc în MySQL, îmbunătățindu-i performanța, fiabilitatea și ușurința de utilizare.

Figură 2.11 - Logo MariaDB

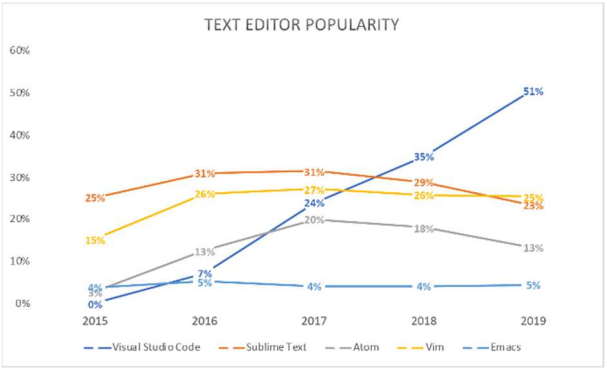
2.8 Visual Studio Code

Visual Studio Code, cunoscut sub numele de VSCode, este un editor de cod gratuit și open-source dezvoltat de Microsoft. Lansat în aprilie 2015, a devenit rapid preferatul dezvoltatorilor datorită caracteristicilor sale puternice și extensibilității. Suportă mai multe limbaje de programare, precum JavaScript, Python și C++, oferind funcții avansate ca evidențierea sintaxei și suport pentru depanare.

VSCode se remarcă prin ecosistemul său extins de extensii, care permit personalizarea mediului de dezvoltare. Include, de asemenea, comenzi Git integrate pentru un control eficient al versiunilor. Editorul este proiectat să fie ușor și rapid, funcționând pe diverse platforme cum ar fi Windows, macOS și Linux. Interfața este foarte personalizabilă, oferind utilizatorilor flexibilitatea de a-și adapta spațiul de lucru după preferințe. Aceste calități fac din VSCode o opțiune populară pentru dezvoltatori din toată lumea.



Figură 2.13 - Logo VSCode



Figură 2.12 - Popularitatea VSCode

2.9 Headless Raspbian

Raspbian fără interfață grafică (headless) este o configurație a sistemului de operare Raspbian, concepută pentru a rula pe dispozitivele Raspberry Pi fără un monitor dedicat, tastatură sau mouse. Această configurare este ideală pentru aplicații de server sau pentru Internet of Things (IoT), unde accesul fizic la dispozitiv este minim sau nu este necesar. Funcționând în modul "fără cap", Raspbian poate fi gestionat de la distanță prin conexiuni de rețea folosind SSH (Secure Shell) sau alte protocoale de acces la distanță.

Această configurație reduce cerințele de hardware și consumul de energie, fiind foarte eficientă pentru proiecte care necesită un sistem ușor, stabil și care să funcționeze continuu. Raspbian fără cap include toate pachetele de software esențiale și caracteristicile găsite în Raspbian standard, dar este administrat de obicei prin interfețe de linie de comandă sau software de desktop la distanță, permițând utilizatorilor să implementeze, să actualizeze și să gestioneze aplicații fără interacțiune directă cu hardware-ul.

Raspbian fără cap este deosebit de popular printre dezvoltatori și pasionați pentru aplicații cum ar fi sistemele de automatizare a locuinței, serverele web și instrumentele de monitorizare a rețelei, unde interfața grafică nu este necesară și resursele sunt utilizate optim.

2.10 Heidi SQL

HeidiSQL este un instrument de gestionare a bazelor de date, eficient și ușor de utilizat, care suportă mai multe sisteme de baze de date, inclusiv MySQL, MariaDB, PostgreSQL și Microsoft SQL Server. Această versatilitate îl face esențial pentru dezvoltatorii web și administratorii de baze de date.

Principalele funcții ale HeidiSQL includ capacitatea de a se conecta la mai multe servere simultan, executarea de interogări SQL, editarea datelor în tabele, și importul/exportul de date în diverse formate. De asemenea, oferă unelte pentru diagnosticarea și optimizarea bazelor de date, toate accesibile printr-o interfață intuitivă.

HeidiSQL este un software open-source, disponibil gratuit, care beneficiază de îmbunătățiri continue datorită contribuțiilor unei comunități active. Aceste caracteristici fac din HeidiSQL o opțiune fiabilă și adaptabilă pentru gestionarea bazelor de date.

3 Construirea Stației Meteo

3.1 Descrierea secțiunii de lucrare

Senzorii menționați anterior sunt implicați în realizarea circuitelor care se află la baza stației meteo. Sunt folosite 2 microcontrolere și fiecare are asociat propriul circuit. Fiecare circuit este contruit pe propria placă de prototipare ce urmează a fi legate și depozitate în carcasă folosită. Unul dintre microcontrolere va fi denumit ca *galben* iar celălalt ca *roșu* pentru a putea face diferența mai ușor.

Primul circuit, cel asociat microcontroler-ului galben, gestionează un senzor de lumină și senzorul APDS-9930. Acești senzori împreună au rolul de a putea determina când este ceață afară și când nu este. Senzorul de lumină folosește ADC-ul microcontroler-ului pentru a interpreta semnalul analogic, iar în paralel, celălalt senzor trimite informațiile folosind interfața I^2C. Acest ansamblu are rolul de a gestiona aspectul de detecție al ceții.

Al doilea circuit, cel asociat microcontroler-ului roșu, gestionează senzorii de temperatură/umiditate, anemometrul și senzorul de presiune atmosferică. Senzorul DHT22 va transmite datele folosind unul din pinii digitali, anemometrul va avea la ieșire un semnal analogic gestionat de ADC, iar senzorul de presiune trimite trimite datele folosind interfața I^2C.

Întregul ansamblu compus dintre cele 2 circuite și RaspberryPi va fi ținut într-o incintă realizată prin intermediul unei imprimante 3D care să faciliteze utilizarea cât mai corectă a senzorilor.

3.2 Prezentarea circuitelor folosite

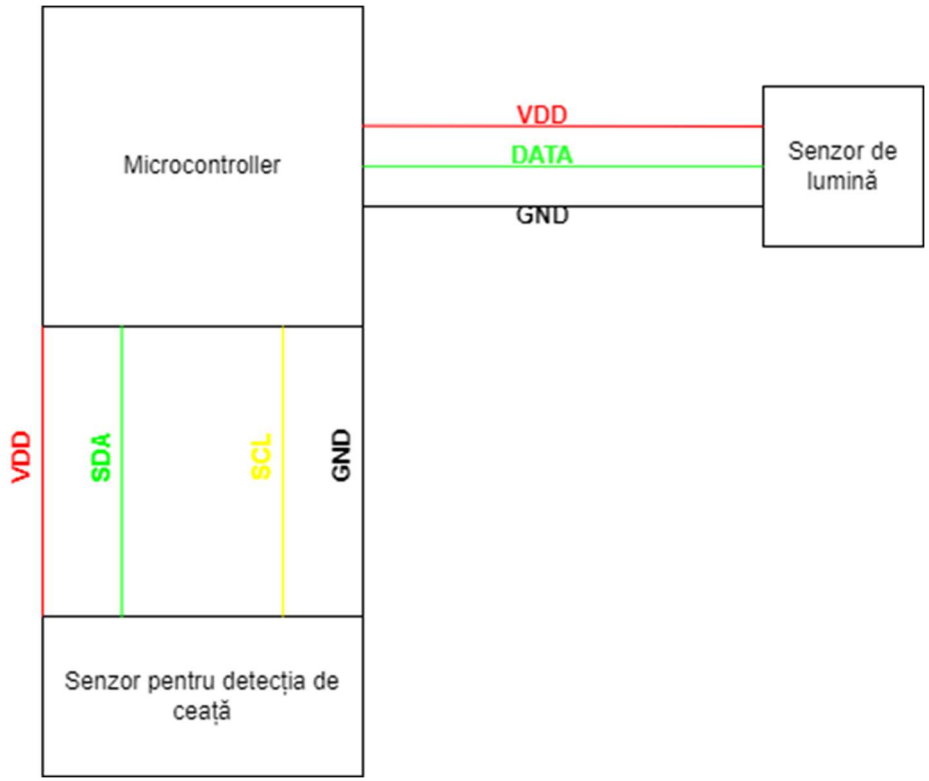
3.2.1 Descrierea Conexiunilor Microcontroler Roșu

Senzorul DHT22 are un singur PIN de date, mai exact **DHT22.DATA**, care va fi legat la PIN-ul **D4** al microcontroller-ului, pe lângă alimentarea la **3.3V** și **GND.** Anemometrul are 2 conexiuni(**SIGNAL** și **GND**), semnalul acestuia fiind gestionat de PIN-ul **D13**, PIN-ul 4 al modulului ADC2 din microcontroler. Senzorul folosit pentru măsurarea presiunii atmosferice(**BMP180)** folosește o interfață I^2C și se leagă la PINii de la modulul I^2C al microcontroler-ului(**SCL** și **SDA**).

3.2.2 Descrierea Conexiunilor Microcontroler Galben

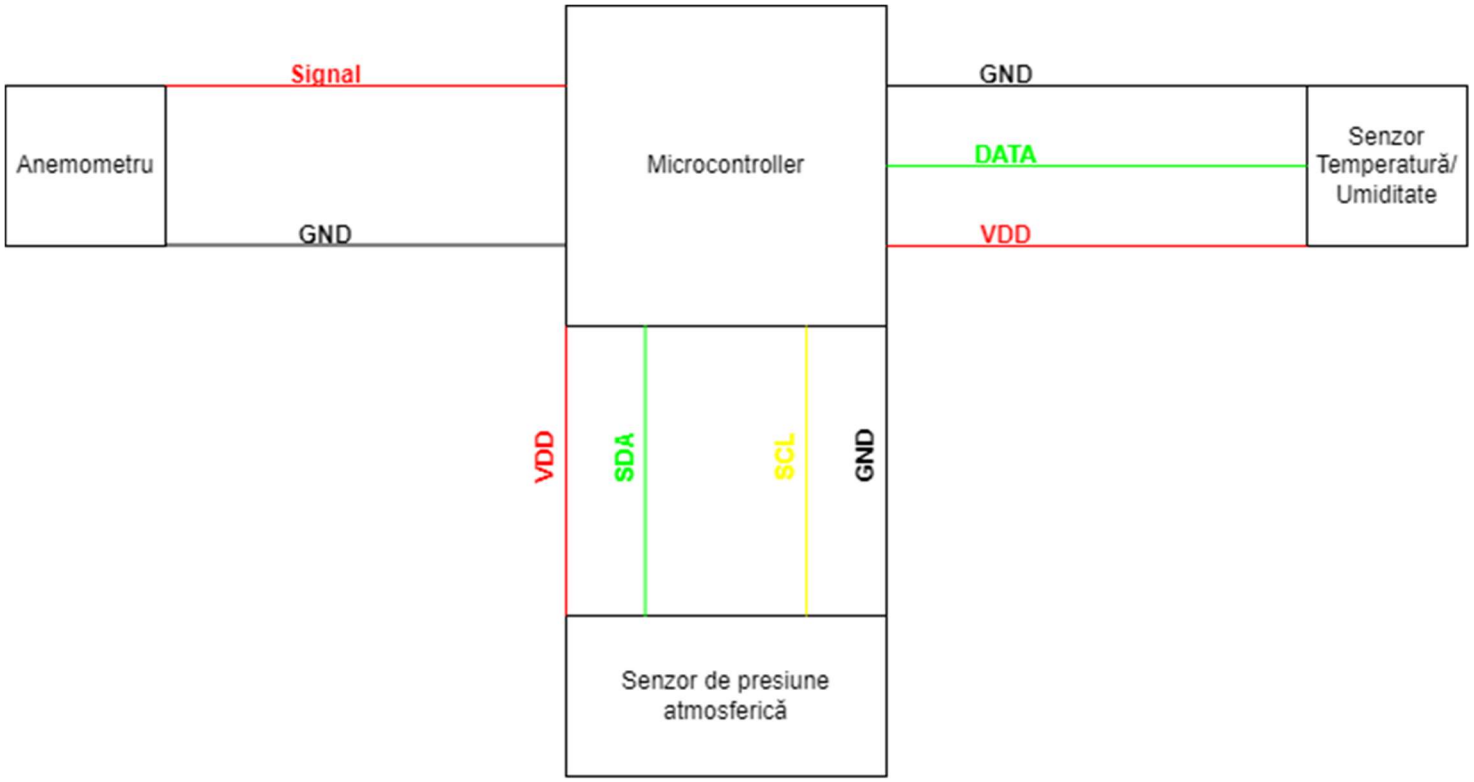
Senzorul de lumină are un output analogic, și este folosit la fel ca și anemometrul, dar legat la **D34**. Senzorul folosit pentru măsurarea distanței cu ajutorul luminii infraroșii (**APDS-9930)** folosește o interfață I^2C și se leagă la PINii de la modulul I^2C al microcontroler-ului(**SCL** și **SDA**).

3.2.3 Schema bloc pentru circuitul microcontroler-ului galben



Figură 3.1- Schema bloc Galben

3.2.4 Schema bloc pentru circuitul microcontroler-ului roșu



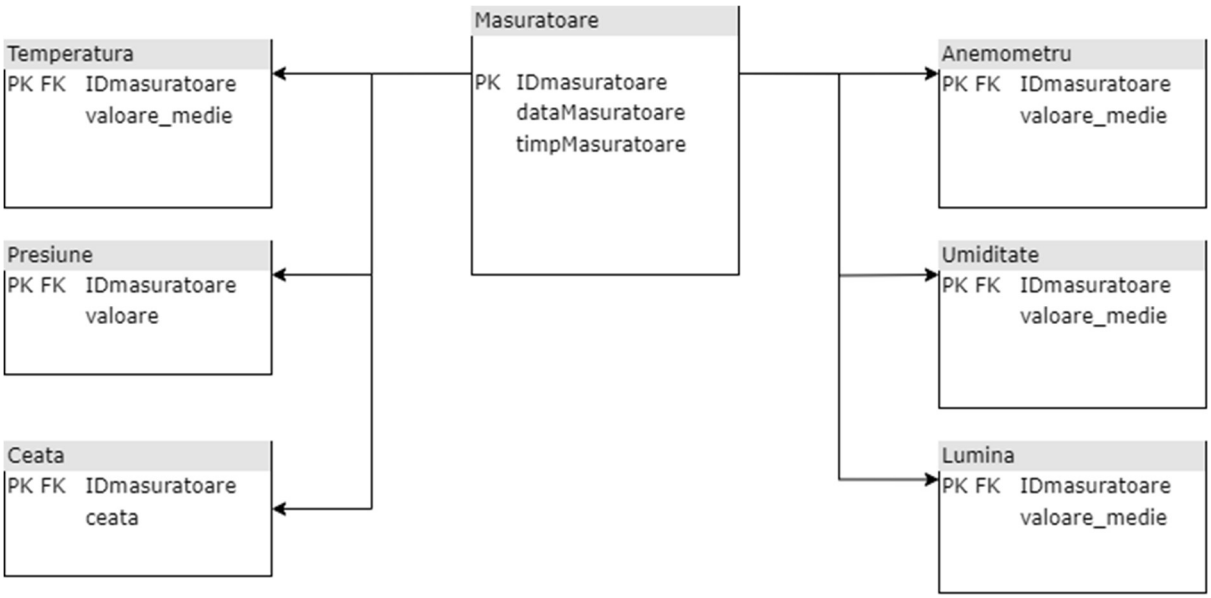
Figură 3.2 - Schema bloc roșu

4 Baza de date locală

4.1 Descrierea secțiunii de lucrare

Datele acumulate de către senzori, după ce au fost trimise către RaspberryPI, vor fi încărcate în baza de date locală a stației meteo, pentru stocare. Ca și sistem de gestionare a bazelor de date se va folosi **MariaDB,** iar interogarile necesare vor fi realizare prin intermediul unui script de Python care se conectează la baza de date și le rulează.

4.2 Schema bazei de date



Figură 4.1 - Schema bazei de date

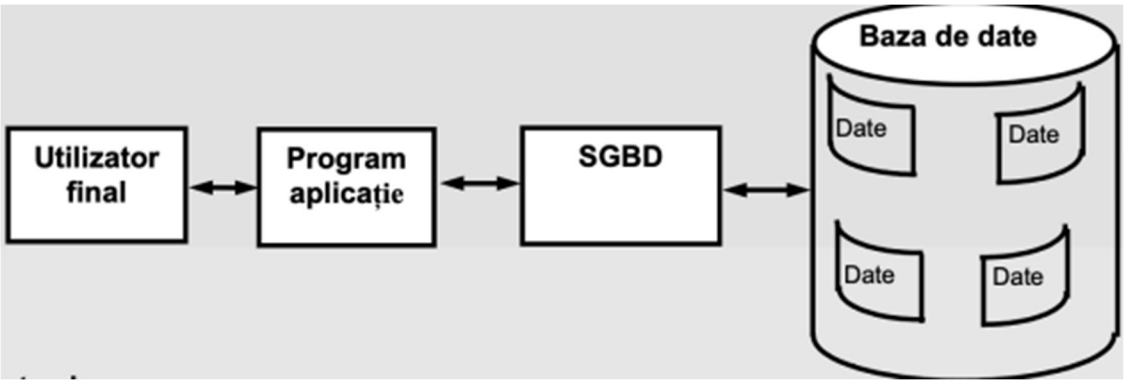
4.3 Elemente de teoria bazelor de date [9]

4.3.1 Aspecte generale despre baze de date

O bază de date este o mulțime de date corelate din punct de vedere logic, care reflectă și modelează un anume aspect al lumii reale, permițând operații de tip CRUD(Create, Read, Update, Delete) asupra datelor.

Componentele principale ale unei baze de date sunt:

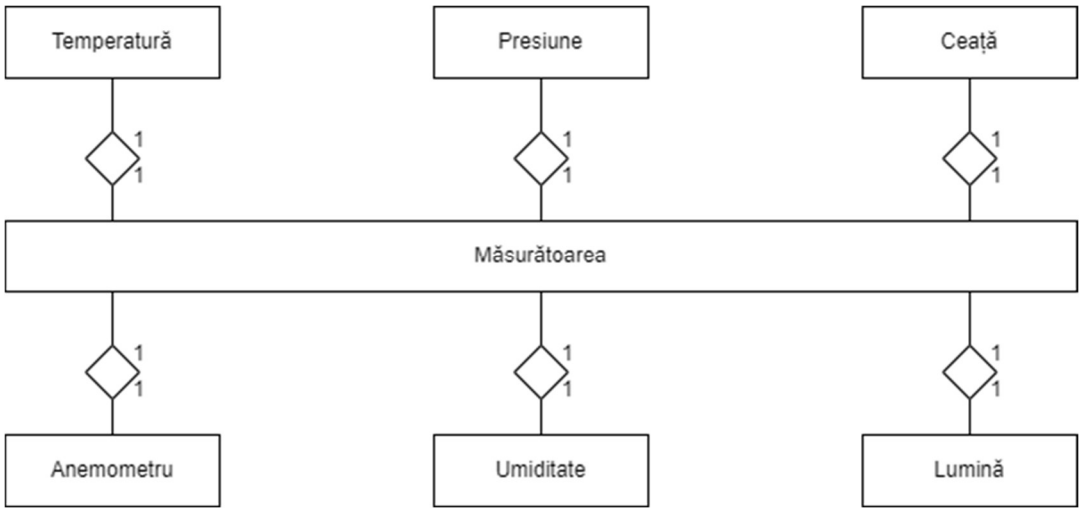
- Componenta hardware – RaspberryPi
- Componenta software – SGDB-ul MariaDB
- Utilizatorii – orice client aflat în rețeaua privată a RaspberryPi-ului
- Date persistente – datele vor fi stocate in memoria internă



Figură 4.2 - Componentele unei baze de date

4.3.2 Cardinalitatea asocierilor

Cardinalitatea unei asocieri reprezintă numărul maxim de entități dintr-o mulțime care pot fi asociate unei singure entitați dintr-o altă mulțime. Pentru baza mea de date cardinalitățile tuturor asocierilor sunt de **1:1.** Acest lucru are rolul de a păstra unicitatea fiecărei măsurători și a avea un set de date potrivit pentru antrenarea modelului.



Figură 4.3 - Diagrama Entitate Asociere

4.4 MariaDB Connector/Python

Connectorul MariaDB este o librărie software care permite comunicația dintre anumite aplicații software și o bază de date MariaDB. Acesta este un API(mai exact un DB-API 2.0 pentru Python) care îi oferă aplicației web abilitatea de a rula comenzi SQL pe baza de date la care s-a realizat conexiunea. Acest connector este optimizat pentru a produce performanțe considerabile și include capabilități cum ar fi compresia datelor și transmiterea datelor pe canale securizate. Connectorul este o unealtă open-source și este complet compatibilă cu atât

cu orice forma de baza de date MariaDB, cât și cu baze de date de tipul MySQL, datorită faptului că MariaDB este un fork de MySQL.

Deoarece lucrez pe un sistem de operare headless, am luat decizia de a rula toate instrucțiunile asociate bazei de date în cadrul unor scripturi de Python, atât pentru a ușura munca cât și pentru a avea o bază software solidă a acestui proiect. Practic, poate fi utilizat orice alt RaspberryPi, atâta timp cât codurile sunt preluate. Există un script pentru inițializarea de la 0 a bazei de date indiferent de stadiul în care se află hardware-ul și un alt script de populare a bazei de date. Ambele scripturi pot fi găsite în sectiunea de **Anexe** sub numele de *create _databse_queries.sql și fill_database.sql.*

5 Noțiuni teoretice pentru învățarea automată

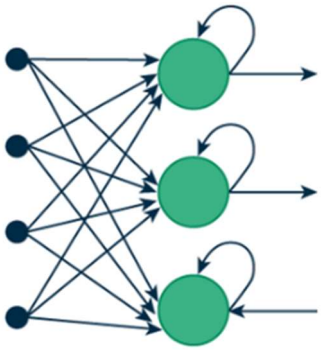
5.1 Descrierea secțiunii de lucrare

În acest capitol al lucrării, voi prezenta algoritmii de învățare automată folosiți de către mine pentru aprecierea și studiul condițiilor meteo, precum și motivația alegeri fiecărui algoritm și cum se comportă pentru diferite seturi de date analizate. Pentru această dezvoltare va fi considerat: Long Short Term Memory, care este o rețea neurală recurentă.

5.2 Rețele neurale recurente

Un algoritm de învățare automată reprezintă un set de reguli și procese folosite de un sistem cu inteligență artificială pentru a descoperi modele în seturi de date sau pentru a prezice modul în care se va desfășura un proces înainte ca acesta să aibă loc [10]. Acești algoritmi pot fi folosiți intr-o gamă largă de aplicații, cum ar fi predicții meteo și recunoașterea imaginilor. [11]

Rețelele Neurale Recurente sunt un tip rețele neurale pentru care ieșirea pasului anterior devine intrare pentru pasul curent, asemănător unei funcții recurente. Aceste rețele obțin acest efect prin introducerea unui strat ascuns, care memorează această informație calculată anterior(Hidden State/Memory State). Astfel, aceste rețele devin indispensabile pentru situații cum ar fi recunoașterea de text(unde este necesară memoria cuvintelor citite anterior), versus o rețea neurală de tip feedforward unde acest efect nu ar putea fi obținut



Figură 5.1 - Diagramă RNN

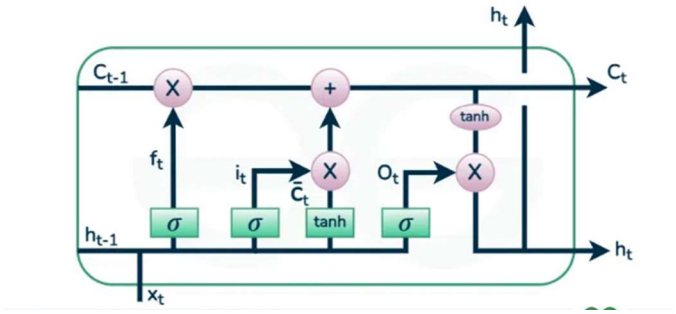
5.3 Long Short -Term Memory

5.3.1 Descrierea generală a rețelei neurale

Rețelele de Memorie pe Termen Lung și Scurt (LSTM) sunt un tip de rețea neurală recurentă (RNN) proiectată pentru a gestiona sarcini de predicție a secvențelor. Spre deosebire de alte rețele neuronale, RNN-urile, și în special LSTM-urile, includ o dimensiune temporală pentru a lua în considerare timpul și secvențele. Acestea sunt recunoscute ca fiind cel mai eficient și proeminent subgrup de RNN-uri, concepute pentru a recunoaște modele în secvențele de date.

Memoria unui LSTM este asigurată de celule de memorie , care au abilitatea de a memora secvențe lungi de date . Fiecare celulă de memorie are 3 componente prin care trebuie să treacă informația: o poartă de intrare, o poartă de *uitare* și o poartă de ieșire. Poarta de intrare decide ce informație se păstrează , poarta de uitare decide ce informație se uită și poarta de ieșire decide formatul informației care urmează să intre în stratul ascuns. [12]

5.3.2 Schema unui LSTM [13]



Figură 5.2 - Schema unui LSTM

5.3.3 Structura unui LTSM

5.3.3.1 Poarta de uitare(Forget gate)

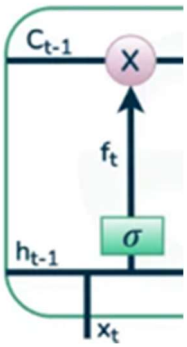
Informația considerată ca ne mai fiind folositoare este eliminată folosind această poartă. Aceasta consideră 2 intrări, mai exact intrarea de la momentul actual($\mathbf{x_t}$) și ieșirea de la celula precedentă($\mathbf{h_{t-1}}$). Aceste componente sunt înmulțite cu matrice de ponderi plus adăugarea unui bias. Rezultatul acestei operații este trecut printr-o funcție sigmoidală ce oferă o ieșire în format binar. Conform acestui rezultat se decide ce celule pot memora informația și ce celule o uită. Ecuația pentru poarta de uitare este:

$$f_t = \sigma(W_f \cdot \{h_{t-1,} x_t\} + b_f)$$

unde:

- W_f este matricea ponderilor
- $\{h_{t-1,} x_t\}$ reprezintă concatenarea dintre intrarea curentă și ieșirea stratului ascuns anterior
- b_f reprezintă biasul porții de uitare

- σ este funcția de activare



Figură 5.3 - Poarta de Uitare

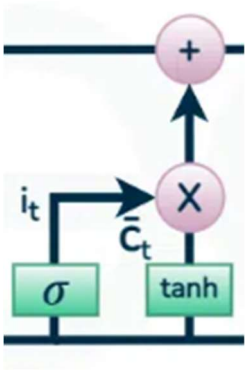
5.3.3.2 Poarta de intrare(Input gate)

Rolul acestei porți este de a adăuga informația utilă. Întâi informația este regulate folosind funcția sigmoidă și filtrate folosind același input ca la poarta de uitare. După, folosind funcția tangent are un rezultat aflat între -1 și 1, care conțin toate valorile posibile de la $\mathbf{h_{t-1}}$ până la $\mathbf{x_t}$. La final, se va face înmulțire între vectorul de ieșire și forma lui regulată pentru a putea obține ieșirea acestei porți. Ecuațiile acestei porți sunt:

$$i_t = \sigma(W_i \cdot \{h_{t-1,} x_t\} + b_i)$$

$$\hat{C}_t = tanh(W_t \cdot \{h_{t-1,} x_t\} + b_t)$$

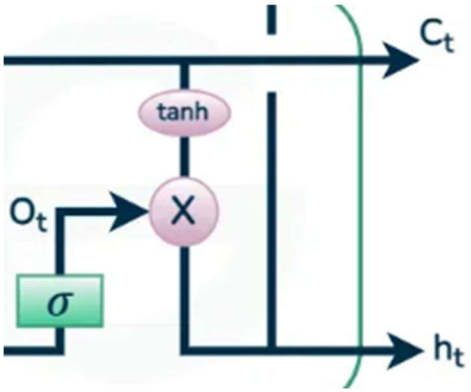
$$C_t = s\, f_t \times C_{t-1} + i_t \times \hat{C}_t$$



Figură 5.4 - Poarta de intrare

5.3.3.3 Poarta de ieșire(Input gate)

Această poarta are rolul de a extrage informațiile utile din starea curentă pentru a fi prezentată ca ieșire a întregului system. Procedeul este asemănator cu cel de la poarta de intrare, doar ca produce 2 ieșire, mai exact stratul curent și stratul ascuns. Diferențierea dintre cele 2 se face prin înmulțirea cu funcția tangentă.



Figură 5.5 - Poarta de ieșire

6 Antrenarea modelului

6.1 Descrierea secțiunii de lucrare

În aceasta parte a lucrării, voi descrie modul în care datele meteorologice acumulate de senzori, completate cu date extrase din baze de date publice meteorologice pentru a antrena algoritmii ce vor fi folosiți în prezicerea condițiilor meteorologice. Așa cum a fost menționat anterior, LSTM care este o rețea neurală recurentă. Antrenarea se va face pe un computer extern, cu putere mare de procesare, iar modelul obținut va fi obținut pentru inferență, care se va realiza local, în timp real, folosind acceleratorul de AI.

6.2 Hardware-ul utilizat

Computerul extern utilizat pentru antrenare este echipat cu o placă video Nvidia Quadro. Acesta este un GPU extrem de util pentru acest scop deoarece oferă compatibilitate cu CUDA, care este platforma de calcul paralel utilizată de placile video de la Nvidia și care este complet compatibilă cu framework-ul Pytorch folosit pentru antrenarea celor 2 modele.

6.3 Setul de date folosit

Pentru antrenarea modelului ce va fi folosit în predicția condițiilor meteorologice am folosit un set de date de pe site-ul *worldweatheronline.com* care acoperă un interval de timp de 5 ani, de la începutul anului 2018 până la începutul anului 2024, înregistrările fiind făcute din oră în oră. Set-ul de date a fost modificat astfel, încât să respecte formatul măsurătorilor făcute de mine, astfel încât să poată fi ușor integrate și utilizate. De asemenea, este foarte important ca datele să fie normalizate, fiind încadrate între valorile 0 și 1, ajutând modelul să ajungă mult mai ușor la convergență.

```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)
```

6.4 Antrenarea Rețelei Neurale

6.4.1 Crearea secvențelor

Un prim pas foarte important pentru antrenarea datelor este crearea de secvențe, care permit algoritmului să învețe din structura temporală a datelor, permițând eficientizarea procesului de predicție. Lucrând cu puncte anteriore de timp(historical data points) și etichetele asociate acestora, rețeaua neurală având astfel capacitatea să observe tendițe în date, cum ar fi schimbările anuale sau sezoniere. Prin ajustarea lungimii secvențelor, algoritmul creștem numărul de seturi de antrenare, fără a a fi nevoie de introducere de date noi. LSTM-urile au memorie, ceea ce înseamnă că pot extrage contextual din aceste diferite seturi de date.

```
def create_sequences(data, sequence_length):
    sequences = []
    labels = []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i+sequence_length])
```

```
        labels.append(data[i+sequence_length])
    return np.array(sequences), np.array(labels)
```

6.4.2 Convertirea în tensori și realizarea setului de date cu tensori

Un tensor repezintă forma generalizată a unei matrice pentru dimensiuni mai mari, acesta fiind practic o matrice n-dimensională. În contextul învățării automate, aceștia reprezintă structura de bază pentru stocarea tuturor datelor de intrare. Sunt foarte eficienți deoarece se pot adapta foarte ușor la orice structură de date și operațiile cu aceștia sunt ușor de paralelizat, ceea ce permite antrenarea lor folosind unități grafice.

Datele meteorologice acumulate, care au fost împățite anterior în secvențe, vor fi modelate în tensori și introduse setul de date al tensorilor(TensorDataset), permițând accesarea valorilor și etichetelor ca făcând parte din același obiect. La final, acest set de date adaptat va fi împărțit în setul de antrenare și setul de test.

```
X_tensor = torch.tensor(X, dtype=torch.float32)
y_tensor = torch.tensor(y, dtype=torch.float32)
dataset = TensorDataset(X_tensor, y_tensor)
```

```
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])
```

6.4.3 Definirea rețelei neurale

Cea mai importantă parte pentru antrenarea cu success a rețelei neurale este definirea potrivită a acestei, bazat pe toate proprietățile specifice unui LSTM. Definirea rețelei este foarte important deoarece este direct legată la abilitatea acesteia de a învăța și a recunoaște tendințe în datele analizate.

Pentru acest scop se va folosi o clasă care reprezintă modelul. Această clasă se extinde din clasa *nn.Module*, care reprezintă clasa de bază pentru orice model antrenat folosind PyTorch. În rest funcționează ca orice altă clasă din Python, folosind un constructor pentru a crea obiectele, în acest caz modelul ce urmează a fi antrenat.

```
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
```

Funcția forward este o altă componentă importantă a acestei clase. Aceasta este o metodă predefinită a clasei *nn.Module*, care în esență include toate calculele ce duc la transformarea stratului de intrare într-un strat de ieșire. În cadrul acestei metode sunt inițializate straturile ascunse(Hidden State) și straturile intermediare(Cell State).

```
 def forward(self, x):
    h_0 = torch.zeros(num_layers, x.size(0), hidden_size).to(device)
```

```
    c_0 = torch.zeros(num_layers, x.size(0), hidden_size).to(device)
```

După executarea acestor funcții, este important să memorăm doar ultimul output, care va reprezenta modelul antrenat.

```
    out, _ = self.lstm(x, (h_0, c_0))
    out = self.fc(out[:, -1, :])
    return out
```

6.4.4 Antrenarea modelului

Pentru a putea antrena modelul este necesar un pas de preprocesare, în care se alege dispozitivul hardware folosit pentru antrenare(unitatea grafică care poate folosi platforma CUDA) și se instanțiază modelul. Fiecare epocă îmbunătățește rezultatul obținut, însă aduce riscul de overfitting.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = LSTMModel(input_size, hidden_size, output_size, num_layers).to(device)
```

De asemenea, trebuie definite funcția de pierdere(Loss Function) și optimizatorul folosit. Funcția de pierdere este folosită pentru acest caz este una bazată pe MSE, care este comună pentru antrenarea unor modele de regresie. Optimizatorul Adam este un algoritm folosit pentru optimzarea parametrilor asociați unor rețele neuronale. Prin această tehnică de optimizare se asigură convergența accelerată a modelului la care se adaugă și abilitatea de a elimina un bias în date.

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

O epocă de antrenare reprezintă o trecere completă prin tot setul de date. Pentru a putea începe antrenarea, modelul trebuie trecut în modul de antrenare ceea ce ne permite să introducem datele în componenta numită DataLoader. După aceste proces, se poate execută o deplasare(Forward Pass), urmată de un calcul al pierderilor

```
outputs = model(sequences)
loss = criterion(outputs, labels)
```

Ultimul pas este reprezentat de utilizarea optimizatorului pentru acest pas, bazat pe propagarea pierderilor înapoi în sistem.

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

6.4.5 Testarea modelului

Testarea modelului este un pas cheie în antrenarea unui model, deoarece putem determina corectitudinea predicțiilor modelului folosind metrici obiective(MSE, MAE și RMSE). Modelul va fi setat în modul de antrenare , iar calculul gradienților va fi oprit , astfel încât vom putea să calculăm parametrii doriți. Această analiză ajută la aprecierea modelului, iar rezultatele pot indica locuri unde modelul poate fi îmbunătățit.

```
model.eval()
```

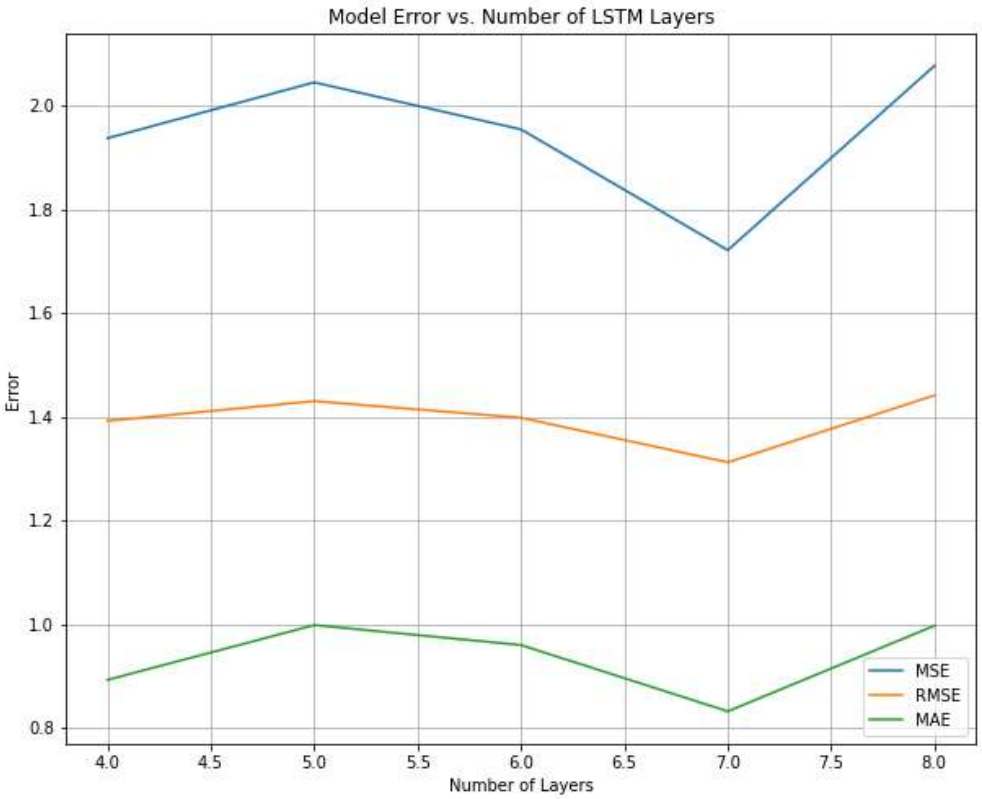
```
test_predictions = []
test_targets = []

with torch.no_grad():
    for sequences, labels in test_loader:
        sequences, labels = sequences.to(device), labels.to(device)
        outputs = model(sequences)
        test_predictions.extend(outputs.cpu().numpy())
        test_targets.extend(labels.cpu().numpy())

test_predictions = np.array(test_predictions)
test_targets = np.array(test_targets)


test_predictions = scaler.inverse_transform(test_predictions)
test_targets = scaler.inverse_transform(test_targets)


mae = mean_absolute_error(test_targets, test_predictions)
mse = mean_squared_error(test_targets, test_predictions)
rmse = np.sqrt(mse)
```

6.4.6 Analiza comparativă a hyperparametrilor

Pentru a putea face o alegere informată cu privire la hyperparametrii de antrenare este important să fie realizată o analiză comparativă acestora pentru a vedea cum se pot obține cele mai bune valori are metricilor (MAE, MSE, RMSE).Parametrii analizați pentru această parte sunt numărul de straturi(num_layer), dimensiunea stratului ascuns(hidden_size), și rata de învățare(learning_rate).

6.4.6.1 Numărul de straturi

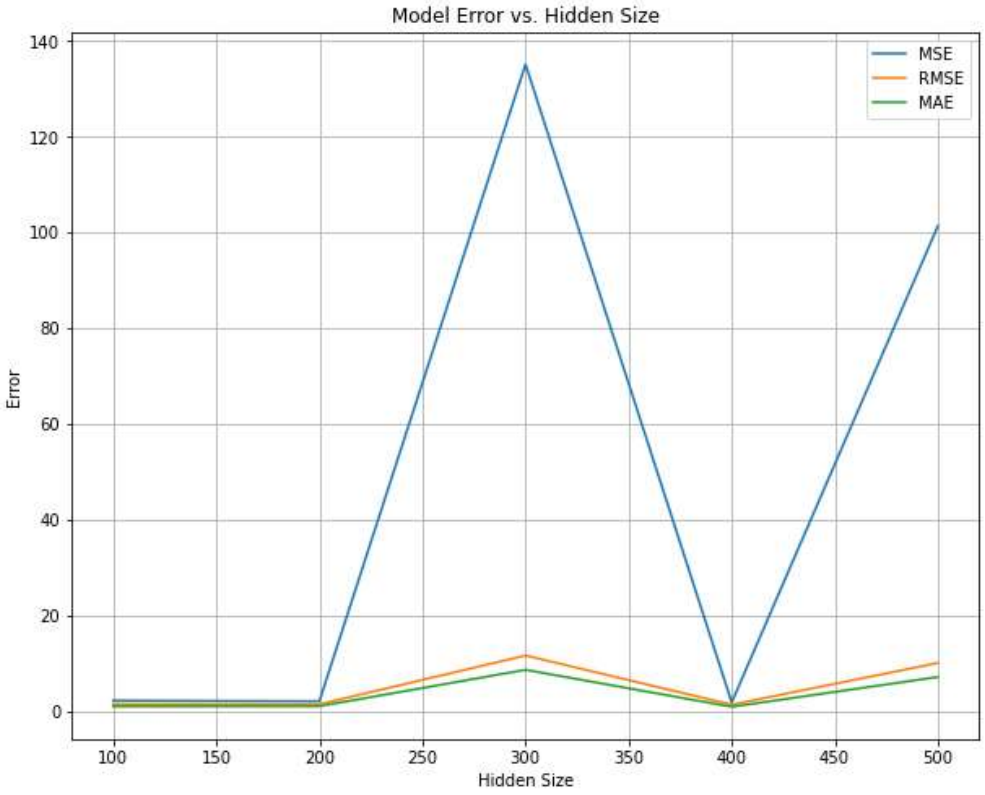
Analiza comparativă pentru numărul de straturi a arătat faptul că valoarea optimă este de 7 straturi, deoarece de la 8 în sus după cum se observă se ajunge la overfitting și valorile metricilor cresc brusc.



Figură 6.1 - Analiză comparativă număr de straturi

6.4.6.2 Dimensiunea stratului ascuns

În urma acestei analize comparative se observă erori foarte mari în jurul valorilor de 300 și 500 de straturi. În același timp, antrenarea cu valori mai mari de 300 duce la acumularea de valori greșite în cadrul modelului, astfel încât am ales valoarea de 200 ca fiind optimă pentru antrenare.



Figură 6.2 - Analiza comparativa dimensiunea stratului ascuns

7 Interfața Web

7.1 Descrierea secțiunii de lucrare

În această parte a lucrării de licență voi elabora despre modul în care a fost realizată pagina de vizualizare a datelor acumulate de stația meteo. Pentru partea de backend voi construi un API folosind framework-ul Django al limajului Python, iar pentru partea de frontend voi folosi stack-ul HTML5, CSS, JavaScript pentru a putea realiza o interfață plăcută vizual și ușor de folosit. Datele meteo vor fi extrase din baza locală folosind API-ul menționat anterior.

7.2 Construirea unui backend folosind Django

Django este un cadru de dezvoltare web de high-level, open-source, scris în Python. Este conceput pentru a ajuta dezvoltatorii să creeze aplicații web securizate, scalabile și ușor de întreținut. Django adoptă filozofia "totul inclus", ceea ce înseamnă că include o gamă largă de funcționalități predefinite, precum un panou de administrare, autentificare a utilizatorilor și un ORM (Mapping Obiect-Relațional) pentru interacțiunile cu baza de date. Acest cadru pune accent pe reutilizabilitatea și capacitatea de a integra ușor componente.

7.2.1 Definirea modelelor

Modelele reprezintă obiecte are căror rol este de a se asocia tabelelor din baza de date și de a permite modificarea și afișarea datelor. Această asociere este cunoscută sub numele de ORM(Object-Relational Mapping), ceea ce înseamnă că interacțiunea cu baza de date se realizează în contextul limbajului Python, nu sub formă de comenzi de MariaDB. Modelele capătă între ele aceleași relații ca și tabelele din baza de date(în cazul meu toate sunt la nivelul one to one). Astfel, interacțiunea cu baza de date este una facilă și ușor de implementat. Modelele se regăsesc în fișierul *models.py,* și mai jos se regăsește structura unuia dintre modelele folosite, codul întreg fiind disponibil în anexe. Acest model gestionează tabelul care stochează datele de temperatură receptate de senzori în variabile de tip float.

```
class Masuratoare(models.Model):
    dataMasuratoare = models.CharField(max_length = 100)
    timpMasuratoare = models.IntegerField()
```

Modelel în Django trebuie migrate, aceasta fiind o metodă prin care mă pot asigura că schema bazei de date se păstrează consistentă indiferent de modificările aduse. Practic, baza de date se modifică fără ca eu sa mai folosesc instrucțiuni de tip SQL.

```
python manage.py makemigrations
python manage.py migrate
```

7.2.2 Implementarea de Views și Serializers

Serializării Django, folosiți în cadrul Django REST Framework (DRF), transformă tipuri de date complexe, cum ar fi queryset-uri și instanțe de modele, în tipuri de date native Python. Acestea pot fi apoi convertite ușor în JSON, XML sau alte tipuri de conținut. Serializatorii permit de asemenea deserializarea, convertind datele parsate înapoi în tipuri

complexe, după ce validează datele de intrare. Acest proces este esențial pentru API-uri, asigurând că datele complexe din modele sunt transformate într-un format accesibil pentru clienți, și invers, că datele primite de la clienți sunt procesate și stocate în siguranță.

```
class MasuratoareSerializer(serializers.ModelSerializer):
    class Meta:
        model = Masuratoare
        fields = '__all__'
```

View-urile Django sunt componente esențiale ale unei aplicații web Django, făcând legătura între datele aplicației și utilizatori. În arhitectura Model-View-Template (MVT) a Django, view-ul gestionează logica, procesând cererile utilizatorilor și returnând răspunsuri. Acestea pot fi HTML, redirecționări, erori 404, JSON, XML sau alte formate. Vederile utilizează modele pentru a accesa datele necesare și delegă formatarea datelor către șabloane. Django oferă flexibilitate prin suportul pentru vederi bazate pe funcții și pe clase, facilitând gestionarea eficientă a fluxurilor de lucru și reutilizarea codului. Pentru aplicația mea , voi folosi view-uri care returneaza JSON ,deoarece acesta este formatul care poate fi utilizat cel mai ușor în relație cu D3.js pentru a genera graficele. Un exemplu de view folosit în acest API este :

```
class TemperaturaViewSet(viewsets.ModelViewSet):
    queryset = temperatura.objects.all()
    serializer_class = TemperaturaSerializer
```

7.2.3 URL-uri în Django

URL-urile Django sunt definite în URLconf (configurația URL), care este o legătură între modele de URL-uri și views. Acest sistem permite Django să gestioneze și să direcționeze cererile utilizatorilor către view-ul potrivită bazată pe URL. URL-urile sunt configurate într-un fișier Python din folder-ul aplicației webapp.

```
router = DefaultRouter()
router.register(r'masuratoare', MasuratoareViewSet)
router.register(r'temperatura', TemperaturaViewSet)
router.register(r'presiune', PresiuneViewSet)
router.register(r'ceata', CeataViewSet)
router.register(r'anemometru', AnemometruViewSet)
router.register(r'umiditate', UmiditateViewSet)
router.register(r'lumina', LuminaViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

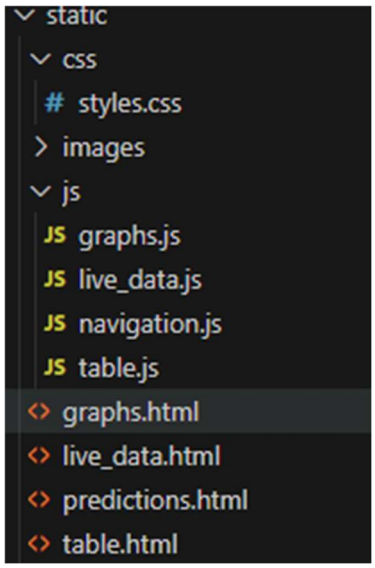
Configurația URL în Django folosește șiruri de caractere care pot include argumente transmise vizualizării asociate. Modelele pot folosi expresii regulate pentru potriviri complexe sau convertoare de cale mai simple. De asemenea, Django suportă gestionarea

spațiilor de nume, facilitând organizarea URL-urilor pe aplicații și utilizarea lor în șabloane fără a le codifica direct.

7.3 Dezvoltare unui frontend folosind HTML, CSS, JavaScript și D3.js

7.3.1 Structura de foldere

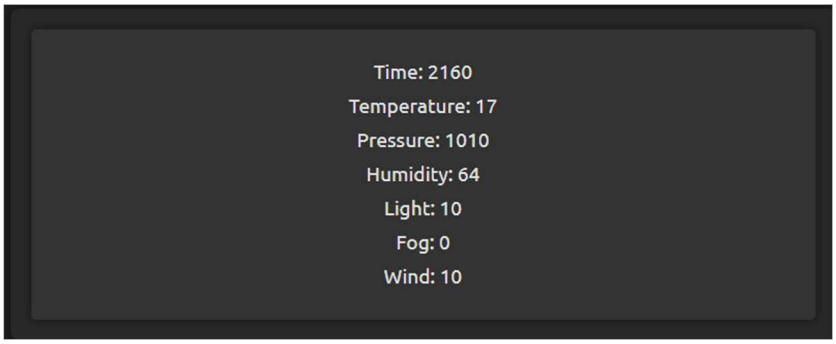
Având un front-end construit peste un backend de Django, există o anumită structură de foldere pe care trebuie să o respect astfel încât să funcționeze corect. Din perspectiva acestui API se definesc 2 tipuri de fișiere asociate componentei de front-end, mai exact fișiere media și fișiere statice. Fișierele statice includ tot ce ține de CSS, Javascript, HTML5 și D3.js, fiind însă asociate unei singure aplicații Django(webapp în contextul lucrării mele).



Figură 7.1 - Structura de Foldere

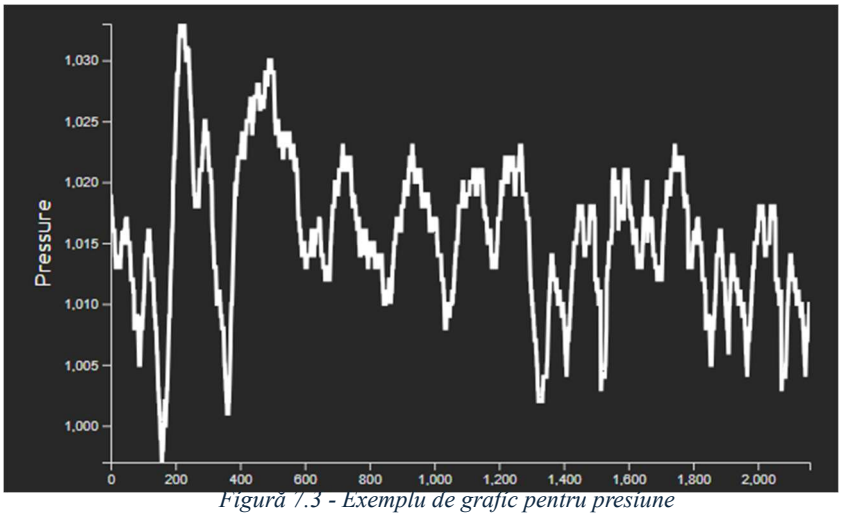
7.3.2 Construcția aplicației

Aplicația constă în 4 meniuri diferite, mai exact meniul de date actuale, meniul de grafice, meniul de predicție și meniul de tabel. În meniul de date curente se va afișa ultima înregistrare din baza de date(din punct de vedere temporal) într-un format ușor de citit și de înțeles.



Figură 7.2 - Meniul de date curente

Meniul de grafice va realiza câte un grafic diferit pentru fiecare parametru de vreme măsurat. Aceste grafice au fost realizate cu ajutorul framework-ului D3,js, acestea fiind complet interactive. De asemenea tot pe această pagina se va calcula și afișa dispersia și media datelor pentru fiecare grafic.



Figură 7.3 - Exemplu de grafic pentru presiune

Meniul de predicții va arată, conform rezultatului dat de model, predicția pentru momentul de timp care se află cu 24 de ore în viitor față de momentul arătat în pagina de date curente, pentru a putea face o comparație. Pentru această predicție există un model separat în backend, care este apelat pentru a putea afișa datele cerute. Ca și format vizual, arată exact la fel ca și pagina cu datele curente.

Ultimul meniu este cel cu tabelul de date. Aici vor fi afișate, sub forma unui tabel, toate datele care sunt disponibile în acel moment în baza de date. Tabelul este obținut prin reunirea tuturor tabelelor care stochează datele meteorologice, excluzând însă tabelul predicțiilor.

Time	Temperature	Pressure	Humidity	Light	Fog	Wind
1	6	1019	82	10	0	8
2	6	1018	83	10	0	8
3	6	1018	84	10	0	7
4	5	1017	85	10	0	6
5	5	1017	85	10	0	5
6	5	1017	85	10	0	5
7	5	1016	85	10	0	4
8	5	1016	85	10	0	4
9	5	1016	85	10	0	5
10	6	1016	83	10	0	6
11	7	1016	76	10	0	7
12	8	1016	69	10	0	9
13	9	1015	64	10	0	11
14	10	1014	61	10	0	13
15	11	1014	58	10	0	15
16	11	1013	59	10	0	14

Figură 7.4 - Exemplu tabel

Pentru a putea naviga între cele 4 pagini, am introdus o bară de navigare, unde fiecare meniu poate fi accesat prin intermediul propriului buton.

Figură 7.5 - Bara de navigare

Concluzii

În urma realizării proiectului am reușit să creez o stație meteorologică care se bazează pe tehnologia IoT. Acest dispozitiv este destinat persoanelor care doresc să capete o perspectivă mai bună asupra modului în care evoluează schimbările climatice. Proiectul combină concepte din toate ariile tehnologiei, permițându-mi să demonstrez cunoașterea acestor domenii prin realizarea dispozitivului.

Relizarea acestei stați meteo folosește atât tehnologii bine documentate, care au fost analizate și dezvoltate de comunitățile lor , dar și tehnologii noi, cum ar fi acceleratoarele hardware, care permit dispozitivelor IoT să atingă praguri noi care erau inaccesibile înainte datorită restricțiilor de consum și hardware. Am ales ca toate tehnologiile software implicate în acest proiect să fie open source, astfel încât proiectul meu să intre sub incidența conceptului de copy-left, devenind accesibil oricui vrea să încerce să îl recreeze.

Complexitatea temei abordate, precum și plaja largă de domenii pe care se întinde a necesitat apelarea la un număr de cunoștiințe teoretice și abilități practice studiate în cei 4 ani ai ciclului de licență:

- Cunoștiințele de arhitectura sistemelor mi-au permis să creez un ansamblu durabil și eficient, alegând cele mai potrivite componente pentru sarcina prezentată și interfețele potrivite care le leagă între ele.
- Programarea de microcontrolere are de a mă ajuta să interpretez cum trebuie semnalele analogice și digitale de la senzori și construirea de librării pentru senzorii care aveau nevoie . De asemenea a fost cheie și înțelegerea fenomenului fizic al fiecărui senzor pentru a putea crea modelul de programare potrivit.
- Formalismele matematice asociate bazelor de date îmi permit să creez un sistem care face sens matematic, acesta căpătând astfel o rigiditate care nu era posibilă de obținut dacă nu țineam cont de rapoartele de cardinalitate și modul cum se leagă tabelele.
- Alegerea algoritmului de învățare automată a fost rezultatul ultimilor doi ani de studiu pe care i-am avut la diferitele cursuri de specialitate în cadrul facultății. Alegerea de rețea neuronală recursivă a fost ușoară, iar framework-ul PyTorch a dovedit că are rezultate mult mai bune din puncte de vedere al timpului de antrenare decât TensorFlow.
- Cunoașterea modului în care funcționează tehnologiile web precum și cele despre interfețe om-mașină mi-au permis să construiesc o interfață ușor de utilizat și care pune foarte bine în evidență datele meteorologice.

Acest proiect are însă și loc de îmbunătățire, având arii în care lucrurile ar putea fi gestionate altfel decâ propunerea mea. Un prim lucru ar fi adăugarea de diferite forme de senzori pentru a explora o serie mult mai mare de fenomene meteorologice. De asemena pentru a reduce timpii de acces si complexitatea acestu ansamblu se poate folosi un singur

microcontroler în loc de două, cu mențiunea că va trebui multiplexată interfața I^2C pentru a putea acomoda ambii senzori. Pe partea de baze de date, putea fi folosită o bază de date nonrelațională, având toate datele în același tabel. Acest lucru ar putea reduce timpul de acces la baza de date.

[14]

Bibliografie

[1] „Running a brushed DC Motor as a Generator,” 12 2021. [Interactiv]. Available: https://www.portescap.com/en/newsroom/whitepapers/2021/12/running-a-brushed-dc-motor-as-a-generator#:~:text=As%20a%20motor%27s%20rotor%20rotates,termed%20the%20%27back%20EMF%27.. [Accesat 28 March 2024].

[2] Avago Technologies, *APDS-9930 Datasheet.*

[3] RaspberryPi Ltd., „raspberrypi.com,” RaspberryPi Ltd., 14 Mai 2024. [Interactiv]. Available: https://www.raspberrypi.com/documentation/accessories/m2-hat-plus.html. [Accesat 5 Iunie 2024].

[4] Hailo, „hailo.ai,” Hailo, [Interactiv]. Available: https://hailo.ai/products/ai-accelerators/hailo-8l-ai-accelerator-for-ai-light-applications/#hailo8l-features. [Accesat 5 Iunie 2024].

[5] „python.org,” [Interactiv]. Available: https://www.python.org/doc/essays/blurb/. [Accesat 15 Aprilie 2024].

[6] „online.wlv.ac.uk,” University of WolverHampton, 31 Mai 2022. [Interactiv]. Available: https://online.wlv.ac.uk/what-is-html5/. [Accesat 15 Aprilie 2024].

[7] „developer.mozilla.org,” [Interactiv]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS. [Accesat 15 Aprilie 2024].

[8] „developer.mozilla.org,” [Interactiv]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Accesat 15 Aprilie 2024].

[9] V. Pupezescu, „Curs de Proiectare al Bazelor de Date,” București, 2024.

[10] IBM, „ibm.com,” IBM, [Interactiv]. Available: https://www.ibm.com/topics/machine-learning-algorithms. [Accesat 5 Iunie 2024].

[11] „GeeksforGeeks,” 15 Noiembrie 2023. [Interactiv]. Available: https://www.geeksforgeeks.org/machine-learning-algorithms/. [Accesat 5 Iunie 2024].

[12] GeeksforGeeks, „GeeksforGeeks,” 10 Iunie 2024. [Interactiv]. Available: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/. [Accesat 12 Iunie 2024].

[13] „GeeksforGeeks/LTSM,” 10 Iunie 2024. [Interactiv]. Available: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/. [Accesat 10 iunie 2024].

[14] G. Tudor-Bogdan(MasterProgramerCPU), „github.com,” [Interactiv]. Available:

] https://github.com/MasterProgramerCPU/Statie-Meteo-Bazat-pe-IoT. [Accesat 27 Iunie 2024].

[15 *Datasheet BMP180,* -: BOSCH, 2015.
]

[16 „What is Piezoelectricity,” 10 05 2023. [Interactiv]. Available:
] https://vajiramandravi.com/upsc-daily-current-affairs/prelims-pointers/what-is-piezoelectricity/. [Accesat 26 Martie 2024].

[17 T. Liu, *Datasheet DHT22,* Aosong Electronics.
]

[18 R. Frank, „sensortips.com,” 16 Ianuarie 2021. [Interactiv]. Available:
] https://www.sensortips.com/featured/what-is-the-difference-between-an-ntc-and-a-ptc-thermistor/. [Accesat 12 Martie 2024].

[19 K. Sulaiman, „ResearchGate,” Noiembrie 2012. [Interactiv]. Available:
] https://www.researchgate.net/figure/Capacitance-versus-relative-humidity-relationship-for-the-Al-VOPcPhO-Au-capacitive-sensor_fig6_234055193. [Accesat 15 Martie 2024].

[20 Raspberry Pi Ltd, *Raspberry pi 5 product brief,* Raspberry Pi Ltd, 2024.
]

[21 A. N. A. N. Z. A. R. Marwa Sattar Hanoon, „nature.com/Developing Machine learning
] algorithms for meteorological temperature and humidity forcasting,” 2021 Septembrie 2021. [Interactiv]. Available: https://www.nature.com/articles/s41598-021-96872-w#:~:text=This%20study%20proposes%20different%20machine,and%20relative%20humidity%20(Rh).. [Accesat 5 Mai 2024].

[22 X.-H. L. L. N. V. S. J. M. Y. G. L. Giang v. Nguyen, *Application of Random Forest*
] *Algorithm for Merging Multiple Satellite Precipitation Products across South Korea,* Remote Sens, 2021.

[23 D. Robinson, „stackoverflow.blog,” 6 Septembrie 2017. [Interactiv]. Available:
] https://stackoverflow.blog/2017/09/06/incredible-growth-python/. [Accesat 15 Aprilie 2024].

[24 „tooploox.com/pytorch vs tensorflow,” [Interactiv]. [Accesat 20 Aprilie 2024].
]

Anexe

1.models.py

```
from django.db import models

class Masuratoare(models.Model):
    dataMasuratoare = models.CharField(max_length = 100)
    timpMasuratoare = models.IntegerField()

class Temperatura(models.Model):
    valoarea_medie = models.IntegerField()

class Presiune(models.Model):
    valoare = models.IntegerField()

class Ceata(models.Model):
    ceata = models.IntegerField()

class Anemometru(models.Model):
    valoarea_medie = models.IntegerField()

class Umiditate(models.Model):
    valoarea_medie = models.IntegerField()

class Lumina(models.Model):
    valoarea_medie = models.IntegerField()

class Predictie(models.Model):
    dataMasuratoare = models.CharField(max_length = 100)
    timpMasuratoare = models.IntegerField()
    valoare_temperatura = models.IntegerField()
    valoare_presiune = models.IntegerField()
    valoare_ceata = models.IntegerField()
    valoare_anemometru = models.IntegerField()
    valoare_umiditate = models.IntegerField()
    valoare_Lumina = models.IntegerField(
```

2.serializer.py

```
from rest_framework import serializers
from .models import Masuratoare, Temperatura, Presiune, Ceata, Anemometru,
Umiditate, Lumina

class MasuratoareSerializer(serializers.ModelSerializer):
    class Meta:
        model = Masuratoare
        fields = '__all__'

class TemperaturaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Temperatura
        fields = '__all__'

class PresiuneSerializer(serializers.ModelSerializer):
    class Meta:
        model = Presiune
        fields = '__all__'

class CeataSerializer(serializers.ModelSerializer):
    class Meta:
        model = Ceata
        fields = '__all__'

class AnemometruSerializer(serializers.ModelSerializer):
    class Meta:
        model = Anemometru
        fields = '__all__'

class UmiditateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Umiditate
        fields = '__all__'

class LuminaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Lumina
        fields = '__all__'
```

3.alegere_num_layer

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the Excel file
file_path = r'C:\Users\tudor\OneDrive\Desktop\facultate\Licenta\antrenare
model\weatherdata1hr2019_2024.xlsx'
data = pd.read_excel(file_path)

# Combine date and time into a single datetime column and set as index
data['datetime'] = pd.to_datetime(data['date'].astype(str) + ' ' +
data['time'].astype(str).str.zfill(4), format='%Y-%m-%d %H%M')
data.set_index('datetime', inplace=True)
data.drop(columns=['Year', 'Month', 'Day', 'time', 'date'], inplace=True)

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Function to create sequences for LSTM
def create_sequences(data, sequence_length):
    sequences, labels = [], []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i + sequence_length])
        labels.append(data[i + sequence_length])
    return np.array(sequences), np.array(labels)

# Preparing data
sequence_length = 60
X, y = create_sequences(scaled_data, sequence_length)
X_tensor = torch.tensor(X, dtype=torch.float32)
y_tensor = torch.tensor(y, dtype=torch.float32)
dataset = TensorDataset(X_tensor, y_tensor)

# Split data into training and testing
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
# Define the LSTM model
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers,
device):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.device = device

    def forward(self, x):
        h_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size,
device=self.device)
        c_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size,
device=self.device)
        out, _ = self.lstm(x, (h_0, c_0))
        out = self.fc(out[:, -1, :])
        return out

# Hyperparameters and device setup
input_size = data.shape[1]
hidden_size = 200
output_size = data.shape[1]
num_epochs = 20
learning_rate = 0.001
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Training and evaluation functions
results = {'num_layers': [], 'MAE': [], 'MSE': [], 'RMSE': []}

def train_and_evaluate(num_layers):
    model = LSTMModel(input_size, hidden_size, output_size, num_layers,
device).to(device)
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    print(f"Starting training with {num_layers} LSTM layers...")

    for epoch in range(num_epochs):
        model.train()
        total_loss = 0
        print(f"Starting Epoch {epoch + 1}/{num_epochs}")
        for sequences, labels in train_loader:
            sequences, labels = sequences.to(device), labels.to(device)
            outputs = model(sequences)
            loss = criterion(outputs, labels)
```

```
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)
        print(f'Epoch [{epoch + 1}/{num_epochs}] Complete - Loss:
{avg_loss:.4f}')

    model.eval()
    test_predictions, test_targets = [], []
    with torch.no_grad():
        for sequences, labels in test_loader:
            sequences, labels = sequences.to(device), labels.to(device)
            outputs = model(sequences)
            test_predictions.extend(outputs.cpu().numpy())
            test_targets.extend(labels.cpu().numpy())

    test_predictions = np.array(test_predictions)
    test_targets = np.array(test_targets)
    test_predictions = scaler.inverse_transform(test_predictions)
    test_targets = scaler.inverse_transform(test_targets)

    mae = mean_absolute_error(test_targets, test_predictions)
    mse = mean_squared_error(test_targets, test_predictions)
    rmse = np.sqrt(mse)
    results['num_layers'].append(num_layers)
    results['MAE'].append(mae)
    results['MSE'].append(mse)
    results['RMSE'].append(rmse)
    print(f'Num Layers: {num_layers}, MAE: {mae:.4f}, MSE: {mse:.4f}, RMSE:
{rmse:.4f}')

for num_layers in range(4, 9):
    train_and_evaluate(num_layers)

# Plotting the results
plt.figure(figsize=(10, 8))
plt.plot(results['num_layers'], results['MSE'], label='MSE')
plt.plot(results['num_layers'], results['RMSE'], label='RMSE')
plt.plot(results['num_layers'], results['MAE'], label='MAE')
plt.xlabel('Number of Layers')
plt.ylabel('Error')
plt.title('Model Error vs. Number of LSTM Layers')
plt.legend()
plt.grid(True)
plt.show()
```

4.alegere_hidden_size

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
import time  # Import the time module

# Load the Excel file
file_path = r'C:\Users\tudor\OneDrive\Desktop\facultate\Licenta\antrenare
model\weatherdata1hr2019_2024.xlsx'
data = pd.read_excel(file_path)

# Combine date and time into a single datetime column and set as index
data['datetime'] = pd.to_datetime(data['date'].astype(str) + ' ' +
data['time'].astype(str).str.zfill(4), format='%Y-%m-%d %H%M')
data.set_index('datetime', inplace=True)
data.drop(columns=['Year', 'Month', 'Day', 'time', 'date'], inplace=True)

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Function to create sequences for LSTM
def create_sequences(data, sequence_length):
    sequences, labels = [], []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i + sequence_length])
        labels.append(data[i + sequence_length])
    return np.array(sequences), np.array(labels)

# Preparing data
sequence_length = 60
X, y = create_sequences(scaled_data, sequence_length)
X_tensor = torch.tensor(X, dtype=torch.float32)
y_tensor = torch.tensor(y, dtype=torch.float32)
dataset = TensorDataset(X_tensor, y_tensor)

# Split data into training and testing
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
# Define the LSTM model
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers,
device):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.device = device

    def forward(self, x):
        h_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size,
device=self.device)
        c_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size,
device=self.device)
        out, _ = self.lstm(x, (h_0, c_0))
        out = self.fc(out[:, -1, :])
        return out
# Hyperparameters and device setup
input_size = data.shape[1]
output_size = data.shape[1]
num_layers = 7
num_epochs = 20
learning_rate = 0.001
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Training and evaluation functions
results = {'hidden_size': [], 'MAE': [], 'MSE': [], 'RMSE': []}

def train_and_evaluate(hidden_size):
    start_time = time.time()  # Start timing
    model = LSTMModel(input_size, hidden_size, output_size, num_layers,
device).to(device)
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    print(f"Starting training with hidden size {hidden_size}...")

    for epoch in range(num_epochs):
        model.train()
        total_loss = 0
        print(f"Starting Epoch {epoch + 1}/{num_epochs}")
        for sequences, labels in train_loader:
            sequences, labels = sequences.to(device), labels.to(device)
            outputs = model(sequences)
```

```
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)
        elapsed_time = time.time() - start_time  # Calculate elapsed time
        print(f'Epoch [{epoch + 1}/{num_epochs}] Complete - Loss:
{avg_loss:.4f} at {elapsed_time:.2f} seconds')

    model.eval()
    test_predictions, test_targets = [], []
    with torch.no_grad():
        for sequences, labels in test_loader:
            sequences, labels = sequences.to(device), labels.to(device)
            outputs = model(sequences)
            test_predictions.extend(outputs.cpu().numpy())
            test_targets.extend(labels.cpu().numpy())

    test_predictions = np.array(test_predictions)
    test_targets = np.array(test_targets)
    test_predictions = scaler.inverse_transform(test_predictions)
    test_targets = scaler.inverse_transform(test_targets)

    mae = mean_absolute_error(test_targets, test_predictions)
    mse = mean_squared_error(test_targets, test_predictions)
    rmse = np.sqrt(mse)
    results['hidden_size'].append(hidden_size)
    results['MAE'].append(mae)
    results['MSE'].append(mse)
    results['RMSE'].append(rmse)
    print(f'Hidden Size: {hidden_size}, MAE: {mae:.4f}, MSE: {mse:.4f}, RMSE:
{rmse:.4f}')

for hidden_size in range(100, 501, 100):
    train_and_evaluate(hidden_size)
# Plotting the results
plt.figure(figsize=(10, 8))
plt.plot(results['hidden_size'], results['MSE'], label='MSE')
plt.plot(results['hidden_size'], results['RMSE'], label='RMSE')
plt.plot(results['hidden_size'], results['MAE'], label='MAE')
plt.xlabel('Hidden Size')
plt.ylabel('Error')
plt.title('Model Error vs. Hidden Size')
plt.legend()
plt.grid(True)
plt.show()
```

4.weather.api/.views.py

```
from django.contrib import admin
from django.urls import path, re_path, include
from django.views.generic import RedirectView
from django.views.static import serve
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('measurements.urls')),
    re_path(r'^$', RedirectView.as_view(url='/static/live_data.html',
permanent=False), name='live_data'),
    re_path(r'^graphs/$', RedirectView.as_view(url='/static/graphs.html',
permanent=False), name='graphs'),
    re_path(r'^predictions/$',
RedirectView.as_view(url='/static/predictions.html', permanent=False),
name='predictions'),
    re_path(r'^table/$', RedirectView.as_view(url='/static/table.html',
permanent=False), name='table'),
    re_path(r'^static/(?P<path>.*)$', serve, {'document_root':
settings.STATICFILES_DIRS[0]}),
]
```

5.measurements/views.py

```
from rest_framework import viewsets
from .models import Masuratoare, Temperatura, Presiune, Ceata, Anemometru,
Umiditate, Lumina
from .serializers import (MasuratoareSerializer, TemperaturaSerializer,
PresiuneSerializer,
                          CeataSerializer, AnemometruSerializer,
UmiditateSerializer, LuminaSerializer)
from rest_framework.response import Response
from rest_framework.decorators import api_view

class MasuratoareViewSet(viewsets.ModelViewSet):
    queryset = Masuratoare.objects.all()
    serializer_class = MasuratoareSerializer

class TemperaturaViewSet(viewsets.ModelViewSet):
    queryset = Temperatura.objects.all()
    serializer_class = TemperaturaSerializer

class PresiuneViewSet(viewsets.ModelViewSet):
    queryset = Presiune.objects.all()
    serializer_class = PresiuneSerializer

class CeataViewSet(viewsets.ModelViewSet):
```

```
    queryset = Ceata.objects.all()
    serializer_class = CeataSerializer

class AnemometruViewSet(viewsets.ModelViewSet):
    queryset = Anemometru.objects.all()
    serializer_class = AnemometruSerializer

class UmiditateViewSet(viewsets.ModelViewSet):
    queryset = Umiditate.objects.all()
    serializer_class = UmiditateSerializer

class LuminaViewSet(viewsets.ModelViewSet):
    queryset = Lumina.objects.all()
    serializer_class = LuminaSerializer
```