МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
Навчально-науковий інститут кібернетики,інформаційних технологій та інженерії
Кафедра комп'ютерних наук та прикладної математики

"До захисту допущений"
Завідувач кафедри
д.т н., проф. Ю.В. Турбал
___ _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

« Розробка системи контролю клімату та моніторингу чадного диму »

Виконав: Ющук Олег Ярославович _____
група ІПЗ-41

Керівник: к.ф.-м.н, доцент, Прищепа О.В. _____

Рівне – 2025

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 52с., 18 рисунків, 8 джерел.

Метою кваліфікаційної роботи є дослідження технологій та розробка системи контролю клімату та моніторингу чадного диму для підвищення безпеки користувачів та покращення умов приміщення.

Об'єкт дослідження: система контролю клімату та моніторингу чадного диму.

Предмет дослідження: технічні та програмні засоби забезпечення контролю параметрів мікроклімату та виявлення чадного диму в приміщенні.

Методи розробки створення даної системи контролю клімату, використовують деякі технології для її побудови такі, як: Django Framework для створення веб-застосунку, Aiogram для створення боту, Django Rest Framework для створення API, SQLite база даних та технології Arduino для створення приладу.

Створено нову систему для безпеки та моніторингу мікроклімату на базі Arduino технологій. Розроблений сайт, використовуючи фреймворк Django, для виведення інформації текстовим форматом та графічно. Розроблений бот, для формування звітів з параметрами мікроклімату. Також бот має можливість писати рекомендації при високій/низькій температурі чи вологості, та надсилає попередження при виявлені небезпеки в виді чадного газу.

Ключові слова: Бот, фреймворк Django, Django Rest Framework, Arduino, пристрій, API, сайт, Aiogram, база даних.

ВСТУП

У сучасному світі технологічний прогрес відкриває широкі можливості: від прогнозування погоди на кілька діб наперед до надточного вимірювання параметрів мікроклімату завдяки спеціальним сенсорам. Метеорологічні станції обладнано численними приладами, які моніторять температуру, атмосферний тиск, вологість повітря, висоту над рівнем моря, а також виявляють небезпечні домішки, здатні негативно впливати на самопочуття.

В даній роботі представлено ідею створення кліматичного моніторингового пристрою, що може бути зібраний вдома. Пристрій дозволяє вимірювати температуру, вологість, тиск та визначати наявність чадного газу. Основними компонентами конструкції є: мікроконтролер Arduino Uno, сенсор MQ-9 для детекції чадного газу та парів спирту, сенсор DHT-11 для вимірювання температури й вологості барометр BMP280, а також KXG1203C - сигналізація. Для візуалізації даних застосовується Telegram-бот та сайт, розроблений на базі фреймворку Django, в якому присутні графіки та сформовані звіти , який надає зручний віддалений доступ до інформації.

Метою цього проекту є розробка та тестування такого пристрою, разом з оцінкою його ефективності у реальному середовищі. Пристрій має потенціал бути корисним у різних галузях: технічних приміщеннях для контролю мікроклімату та виявлення шкідливих газів, а також у побуті - для підтримання комфортного мікроклімату в приватних будинках.

У роботі розглянуто етапи проектування пристрою, аналіз вимог до системи, втілення функціональності, тестування, а також проаналізовано переваги, недоліки та потенціал вдосконалення.

Результати дослідження можуть бути актуальними для широкого кола зацікавлених осіб, зокрема для спеціалістів, які працюють з хімічними речовинами, та для власників житла, які прагнуть забезпечити ефективний контроль мікроклімату у своєму помешканні.

РОЗДІЛ 1

ОГЛЯД ПРОБЛЕМИ МОНІТОРИНГУ ШКІДЛИВИХ РЕЧОВИН

1.1. Розвиток метеорологічних досліджень клімату

Розробка та використання метеорологічних пристроїв стала необхідною ще в античному світі, відомий грецький філософ в своїй праці «Метеорологіка» досліджував дані питання, та тільки в 18 ст. метеорологія стала самостійною як наука [1].

Найбільший розвиток даний напрямок сягають своїм корінням з середини 17 століття, коли вчені та винахідники почали шукати ефективні способи вимірювання кліматичних показників. Першими станціями були відкриті впродовж 1780-1795 роках, які побудували 30 метеостанцій в різних країнах, таких як США та деякі країни Європи .

Використовуються такі основні прилади для вимірювання даних: Термометр, гігрометр, барометр, флюгер, опадомір та анемометр, для дослідження температури, вологості, тиску, напрямок вітру, вимірювання кількості опадів та для вимірювання швидкості вітру[2].

Основне завдання станцій є на базі досліджених даних вивести та спрогнозувати атмосферні явища на декілька днів вперед. Прогнозування навіть на тиждень є тяжкою працею, так як навіть десяті та соті числа в даних температури чи вологості, дає абсолютну інший прогноз погоди.

Завдяки швидкому розвитку технологій сучасні метеорологічні станції оснащені численними інтелектуальними функціями. Вони можуть підключатися до смартфона для зберігання та аналізу даних, використовувати штучний інтелект для підвищення точності вимірювань та мати інші інноваційні можливості.

Зосереджуючись на темі цієї кваліфікаційної роботи, важливо розглянути можливості та переваги дослідження мікроклімату для домашнього використання: Моніторинг мікроклімату на базі Arduino Uno, датчика парів MQ-9, вимірник температури та вологості DHT11, датчик тиску BMP280 та сигналізацію KGX1203C можна використовувати для моніторингу. Ефективний і доступний пристрій для моніторингу.

1.2. Огляд існуючих моніторингових пристроїв

Системи контролю клімату та моніторингу чадного диму широко застосовуються в домах, виробництві, агросекторі та системах "розумного будинку". Наведемо деякий огляд найбільш розповсюджених технологій, що використовуються для таких задач, зокрема, їх переваги та недоліки.

Для комерційних пристроїв (детектори чадного газу + датчики температури/вологості) можна визначити наступні переваги: простота монтажу та використання; наявність сертифікатів та відповідність нормам безпеки, що є обов'язковою умовою; вбудовані дисплеї та звукові сповіщення; енергонезалежні. Проте присутні і недоліки: програмне забезпечення закритого типу; висока ціна при збільшенні обсягів; обмежені можливості зберігання даних в історії або їх передачі онлайн.

Прикладами таких технологій є: Nest Protect, Kidde, Honeywell CO Detectors – комбіновані CO-детектори; Xiaomi Home Sensors, що відстежують температуру, вологість, CO2.

Проєкти на базі Arduino мають певний спектр переваг: повний контроль над процесом функціонування; недороге масштабування; великий вибір сенсорів та методів передавання даних. Серед недоліків можна зазначити: необхідні технічні знання(електроніка,програмування); зменшена надійність без тестування. Найпоширенішими сенсорами є: MQ-9 –сенсор для CO і горючих газів;DHT22, BME280, SHT31 – температурно-вологісні датчики; ESP32 –

мікроконтролер з Wi-Fi/Bluetooth; Arduino/Raspberry Pi – для складніших проєктів з базою даних і сервером.

Також в є ще один варіант – промислові IoT-рішення в яких є наступні переваги: висока точність та надійність; централізований моніторинг сотень датчиків; сертифікація для небезпечних середовищ (виробництво, ферми тощо). Серед недоліків можна невести: велика вартість; закриті протоколи та програмування; обмежені можливості зміни або неможливість масштабування; спрямовані на великий бізнес(фабрики, технічні приміщення, склади, тощо). Наведемо деякі приклади моніторингових систем: Bosch ClimoSiemens Desigo, Sensirion Environmental Nodes.

1.3. Роль Arduino в створенні моніторингових систем

Arduino – мікроконтролер для створення власних приладів таких як: Домашній алкотестер, пристрій для моніторингу погоди, пристрій для безпеки дому, машинка на радіоуправлінні.

Arduino створена саме для різних пристроїв, його використовують, як комп'ютер, та можна запрограмувати на різні випадки життя.

Arduino – це не тільки про мікроконтролер, а також про програмування та розробку. Arduino IDE – це відкрита платформа для програмування мікроконтролерів. В цьому середовищі розробки є буквально все, всі бібліотеки, для всіх датчиків, які створили люди для мікроконтролерів.

Щоб запрограмувати мікроконтролер Arduino Uno використовуються різні мови програмування. В Arduino IDE використовуються найпопулярніші мови програмування це Assembler, C та спрощена версія C++, яка набагато полегшує роботу з мікроконтролерами, та створює менший поріг входу в цю сферу, за рахунок величезної кількості функцій та готових бібліотек для використання.

Отже, Arduino – сильний інструмент для розробки різних пристроїв. В основі роботи покладено саме цей ця технологія так, як він має багато переваг: гнучкість, бібліотеки, інтеграції з іншими платформами, масштабованість та інше. За допомогою цього, з'являється можливість для створення різних пристроїв, саме через цю технологію і створюються нові інноваційні прилади, які полегшують життя людей та збільшують її безпеку.

РОЗДІЛ 2
СИСТЕМА КОНТРОЛЮ КЛІМАТУ ТА МОНІТОРИНГУ ЧАДНОГО ДИМУ

2.1. Архітектура проєкту

Створення архітектури проєкту – є найважливішим кроком при початку роботи над новою системою в світі програмування. На старті проєкту, вирішуються які основні принципи програмування, які фреймворки та на яких мовах програмування буде створений сам проєкт. Для створення системи в даній роботі будуть використовуватися наступні технології:

1) Фреймворк Django – сильний інструмент для створення веб-додатків на мові програмування Python. Чому використаний саме цей фреймворк? Величезна кількість бібліотек та фреймворків для використання та масштабування проєкту, легкість в розумінні та створенні проєкту.

2) Django Rest Framework – даний фреймворк використовується при створенні API для спілкування бази даних та користувацької частини сайту, зручно для використання та створення ендпоїнтів для отримання інформації з БД.

3) Бібліотека Aiogram – це асинхронна бібліотека Python для розробки ботів, що використовує asyncio для високої продуктивності та масштабованості. З її допомогою можна створювати сучасних ботів з ефективною обробкою запитів, що важливо при значній кількості користувачів.

4) Arduino – потужний інструмент для створення різноманітних приладів. Висока масштабованість, легкість змінення деяких датчиків, бюджетність в питанні ціни та зручна в транспортації.

5) База даних SQLite – При створенні продукту, часто використовуються бази даних. SQLite – класична реляційна локальна база даних, проста в використанні та створенні моделей та збереженні даних.

Проєкт створений наступним чином (рис. 2.1):

1) датчики вимірюють на постійній основі кліматичні показники;

2) Arduino зчитує дані з датчиків та формує зручний звіт в форматі json, також якщо високий вміст газу, включається сигналізація;
3) Output.json зчитує порт до якого підключений Arduino та зберігає звіти допоки бот не подасть запит на останні дані;
4) бот по команді або по фоновій задачі формує звіт для користувача, видає попередження, якщо занадто висока/низька температура або вміст речовин в повітрі;
5) після запиту бота до json-файлу, сформовані звіти зберігаються в базі даних через Django Rest Framework;
6) сайт на Django для формування історії звітів та графіків, отримує дані через Django Rest Framework, та формує потрібні графіки та інші функції.

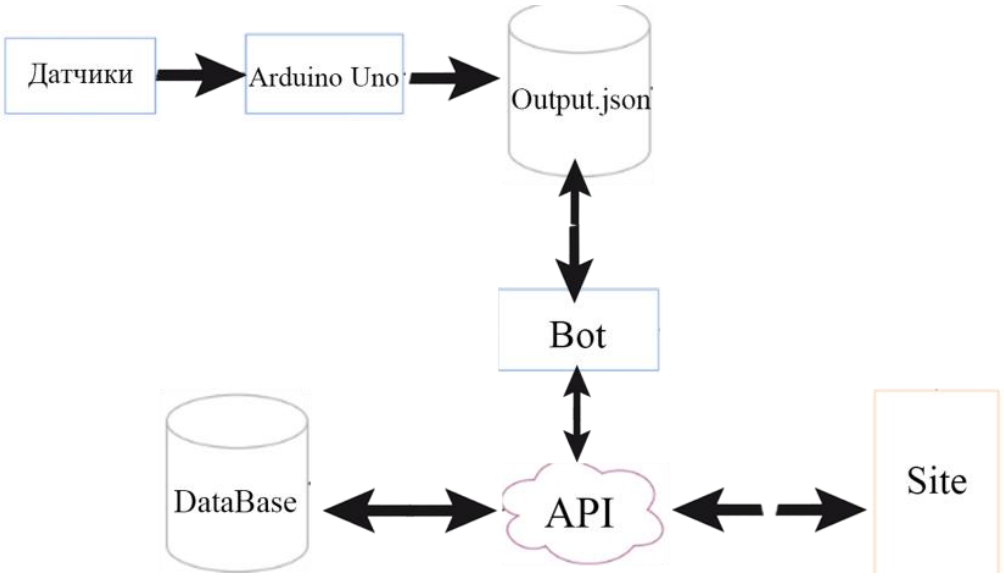


Рис. 2.1. Блок-схема проєкту

2.2. Датчики і мікроконтролери для систем контролю

Найкращий мікроконтролер який можливо тільки купити, для створення власних приладів таких як: Домашній алкотестер, пристрій для моніторингу погоди, пристрій для безпеки дому, машинка на радіоуправлінні – це власне саме про Arduino Uno (рис. 2.2).

Рис. 2.2. Arduino mega 2560

Arduino Uno – це перша частина домашнього моніторингового пристрою, це фундамент, на якому буде будуватися інші частини частини пристрою, для дослідження кліматичних даних.

Найперше, що випало на думку, при виборі мікроконтролеру для пристрою, це надійність та можливість підключити різні датчики та модулі для створення свого проекту. Так як в ньому присутнє все, що потрібно: піни, порти, спосіб підключення, аналогові входи, цифрові входи та інше, завдяки цьому, він ідеально підходить для розробки системи моніторингу.

Характеристикою Arduino Uno є наступною:

1) Мікроконтролер ATmega328
2) Спосіб підключення USB: конвертер CH340
3) Робоча напруга контролера: 5В
4) Порт USB 5В
5) Вхід VCC 5В
6) Вхід Vin 7.5V-12V
7) Цифрові входи/виходи 14 (6 ШІМ виходів)

8) Аналогові входи 6
9) Інтерфейси I2C/TWI, SPI, PWM [3]

Друга частина пристрою – це датчик парів спирту та чадного диму MQ-9 (рис. 2.3).

Рис. 2.3. Датчик MQ-9

Є можливість використання датчика для різних ситуацій, де потрібно виміряти кількість спирту або чадного газу. Також, цей датчик чутливий до парів бензину, що відкриває ще більше можливостей для його використання.

Датчик спроектований на основі двох частин: газоаналізатор MQ-3 та компаратора LM393. Сам газоаналізатор MQ-3 є нагрівальним елементом, який покритий спеціальним напиленням для виявлення спирту в повітрі. При великих концентраціях спирту, в датчику відбувається реакція, яка передає інформацію про концентрацію спирту в диханні. Компаратор – це є перетворювач, який перетворює імпульси датчика в аналогові або цифрові сигнали для виводу результату.

Також, на платі присутні 2 світлодіоди. Один світлодіод (PWR) спалахує коли, підключається живлення до самого датчика, та другий коли концентрація спирту або чадного газу в навколишньому середовищі перевищує допустимі значення.

Датчик підключається до мікроконтролера Arduino mega 2560 за допомогою чотирьох виводів. Подача живлення напругою 5V на датчик

використовується контакт VCC та заземлення GND. Передача результатів йде за допомогою аналогового піна A0 або цифрового піна B0.

Звісно, як і в кожному датчику потрібно пройти калібрування. Для цього потрібно запустити модуль, та протримати його включеним протягом пару годин. Також для калібрування потрібно зафіксувати максимально допустимі дані та мінімальні допустимі дані датчика, та прописати їх в коді програми.

Характеристиками датчика MQ-9 є наступні:

1) напруга живлення: 5 В;
2) споживаний струм: 150 мА;
3) діапазон вимірів: 20ppm – 2000ppm;
4) час прогріву при включенні: 1 хв;
5) вихідний сигнал: аналоговий;
6) робоча температура: -20...+50°C. [4]

DHT-11 – цифровий вимірник, який використовується для вимірювання температури та вологості в повітрі.

Цей модуль є досить популярний в сфері програмування на Arduino, так як видає досить точні дані, які дають можливості для точних розрахунків та вимірів (рис. 2.4).

Рис. 2.4. DHT-11

Складається модуль з датчика вологості та термістора, також містить в собі аналоговий-цифровий перетворювач для аналогових значень вологості та температури.

Для підключення датчику використовуються наступні виходи: VCC; цифровий вихід; заземлення.

Характеристики датчику DHT11 є наступні:

1) модуль: ASAIR DHT11;
2) вологість та допустима похибка: 5 - 95% RH ± 5% (макс.);
3) температура та допустима похибка: -20 ~ +60 ºC ± 2% (макс.);
4) живлення: 3.5-5.5 В;
5) частота: не більше 1 Гц. [5]

BMP280 є найголовнішим датчиком для моніторингу клімату, так як він вимірює відразу 3 показники: Температуру, тиск та висота над рівнем моря.

Цей датчик – це краща версія BMP180. Оскільки в новій версії вимірника такі переваги: менше енергоживлення, висока точність вимірів, точна заводська калібровка та є наявність інтерфейсів I2C и SPI (рис. 2.5).

Рис. 2.5. BMP280

Для підключення в датчика до мікроконтроллера використовуються наступні виходи: VCC; GND; SCL; SDA; CSB; SDO

Arduino Uno має окремі входи для SCL та SDA, тому це ідеальний симбіоз датчика та плати в яку ми будемо підключати датчик.

Характеристиками датчика BMP280 є наступними:

1) напруга: 1.71-3.6В ;
2) інтерфейс: I2C, SPI;
3) калібровка: Заводська;
4) рівень шуму: до 0.2 Па (1.7 см) и 0.01 t;
5) діапазон тиску: от 300 hPa до 1100 hPa (9000 м до -500 м). [6]

Звісно для пристрою, який вимірює чадний дим та інші гази, потрібна також і сигналізація, для цього використовується активний зумер KXG1203C. Невеликий датчик, але потужно випромінює звук, який можна почути за декілька приміщень (рис. 2.6).

Рис. 2.6. KXG1203C

Для підключення використовується два входи: VCC та цифровий вхід. Досить просто настроюється для використання безпосередньо в пристрою.

Характеристики модуля KXG1203C є наступними:

1) макс. частота: 2,7 кГц
2) мін. робоча температура: -30 °C

3) макс. робоча температура: 70 °C
4) мін. напруга живлення: 2 V
5) макс. напруга живлення: 5 V
6) рівень звуку: 85 ДБ
7) діаметр: 12 мм
8) висота: 9,5 мм [7]

2.3. Підключення пристрою

Моніторинговий пристрій мікроклімату з використанням мікроконтролера Arduino Uno, датчика парів спирту та чадного газу MQ-9, датчик температури і вологості DHT-11, BMP280 датчик тиску та активний зумер, який виконує функцію сигналізації KGX1203C є складною системою, і для ефективної роботи всі компоненти повинні бути правильно підключені.

Підключення потрібно для реалізації зручного використання та компактності пристрою (див. рис. 2.7).

1) В базі пристрою використовується Arduino Uno, так як розповідалось в минулих розділах його переваги над іншими можливими пристроями.

2) При підключенні BMP280 потрібно впевнитись, що підключення до живлення відбувається правильно тому, що, якщо підключити до 5+ вольт, датчик може вийти з ладу.

Підключаємо VCC до 3.3 вольт, GND до заземлення та SCL і SDA до окремих виведених входів на платі Arduino Uno.

3) DHT-11 є досить не тяжким в підключенні, але потрібно бути обережним, при підключенні датчика, і впевнитись що він коректно, та немає ніяких збоїв, тому що на моєму досвіді, один датчик поводив себе некоректно, та пошкодився, після чого, потрібна була заміна. Потрібно бути обережним та дотримуватися правил при підключенні.

Підключаємо VCC до 5V, цифровий вихід до піна D4, та заземлення до GND.

4) Датчик MQ-9 є простим в підключенні датчик, і немає ніяких складностей з його під'єднанням.

Підключаємо VCC до 5V, цифровий вихід до піну D2, аналоговий вихід до A0 та заземлення до GND.

5) Зумер KGX1203C є надлегким в підключенні, так як має тільки два виходи: 3v3 живлення та підключення до цифрового входу D7.

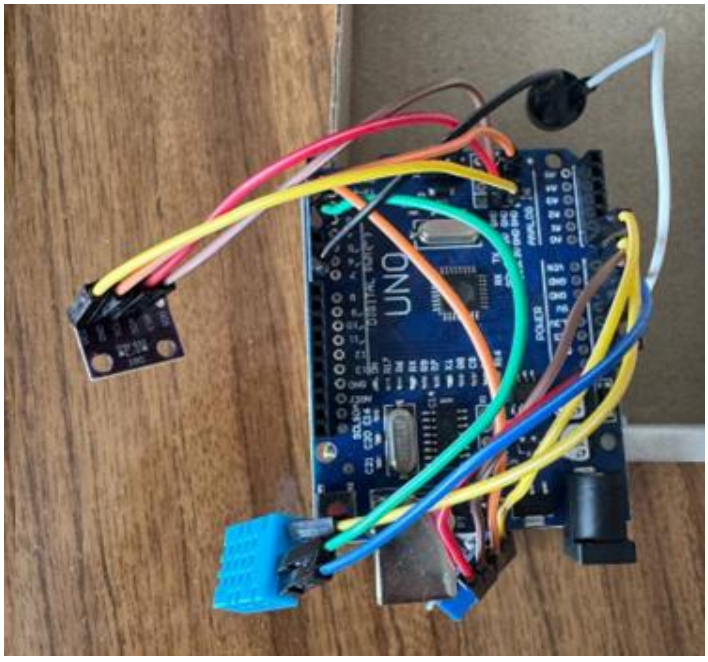


Рис. 2.7. Підключений пристрій

2.4. Програмування пристрою

Для того, щоб написати код для моніторингу, будемо використовувати Arduino IDE, для написання проекту та вивантаження його в пристрій. Для початку, повинні встановити бібліотеку, під назвою “DHT11.h” для того, щоб ми могли взаємодіяти з датчиком DHT11 і також “Adafruit_BMP280.h” для взаємодії

з BMP280. Встановити їх можливо прямо з середовища розробки: Sketch→library→add library→library control. Після встановлення потрібних нам бібліотек, ми можемо почати розробляти наш проект.

```
#include <SPI.h>
#include <Adafruit_BMP280.h>
#include <DHT11.h>
Adafruit_BMP280 bmp; // I2C
#define ledPin 2
DHT11 dht11(4);
int buzzerPin = 7;
float sensorValue;
void setup() {
  Serial.begin(9600);
  pinMode(buzzerPin, OUTPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  if (!bmp.begin(0x76)) {
    Serial.println(F("Не знайдено BMP280!"));
    while (1);
  }
  Serial.println("MQ9 Heating Up!");
  delay(10000);
}
```

Setup-частина проєкту – це встановлення з'єднання та перевірка чи все корректно підключено. Наведемо частину коду setup-частину

#include <SPI.h>, #include <Adafruit_BMP280.h>, #include <DHT11.h> // для коректної роботи програми з датчиками та в майбутньому для вивантаження коду в наш пристрій, підключимо бібліотеки:

int buzzerPin = 7; // Ініціалізуємо пін для роботи сигналізації KGX1203C.

DHT11 dht11(4); // визначимо пін для роботи датчику вологості та температури

float sensorValue; // створимо змінну для датчику газу для виводу з неї даних

unsigned status; status = bmp.begin(0x76); // створюємо змінну для визначення адреси датчику BMP280, та вводимо адрес датчику, в нашому випадку це 0x76**,** для визначення адреси, використовуються спеціальний I2C детектор-

Serial.println("MQ9 Heating Up!"); delay(20000); // для того, щоб датчик mq-9 працював коректно та за максимальною точністю, даємо 20 сек для розігріву елементу датчику.

Переходимо до loop-частини, в якій відбуваються основні процеси та виведення інформації з датчиків.

```
//Data DHT11
  int humidity = dht11.readHumidity();
  int temperature = dht11.readTemperature();
```

int humidity = dht11.readHumidity(); // ініціалізуємо змінну, яка буде зчитувати вологість з датчику DHT11, використовуючи функції з раніше імпортованої бібліотеки <DHT11.h>.

int temperature= dht11.readHumidity(); // ініціалізуємо змінну, яка буде зчитувати температуру з датчику DHT11. Також, як і в минулій змінній, використовуємо метод з бібліотеки <DHT11.h>.

Так як ми ініціалізували датчик BMP280 та підключили його кодом до плати, потрібно просто вивести інформацію з вимірника за допомогою допоміжних функцій з бібліотеки Adafruit_BMP280.h : readTemperature(), readPressure(), readAltitude().

```
//Data BMP280
  float bmpTemp = bmp.readTemperature();
  float pressure = bmp.readPressure() / 100 + 24 ;
  float altitude = bmp.readAltitude(1013.25);
```

Наступний і останній датчик з якого потрібно зчитати дані - це газоаналізатор MQ-9, переініціалізовуємо змінну sensorValue для читання аналогового входу до якого підключений датчик це A0, виводимо значення в Serial.print(). Створюємо if-блок, для перевірки даних, та якщо значення більше 250, світиться розпочинається сигналізація від зумера KGX1203C.

```
// Data MQ-9
 sensorValue = analogRead(A0);

 if (sensorValue < 250) {
  digitalWrite(buzzerPin, HIGH);
  analogWrite(ledPin, outputValue);
 } else {
  digitalWrite(buzzerPin, LOW);
  digitalWrite(ledPin, LOW);
 }
```

Для правильного виведення інформації, потрібно сформувати коректно дані. Так як дані будуть записуватись в json файл, формуємо в відповідному стилі:

```
}
humidity: data
temperature:data
pressure:data
altitude:data
mq:data
{
```

Для кращого та точнішого вимірювання даних, зчитуємо їх кожні 2 секунди.

```
Serial.println(
 String("{\"humidity\":") + humidity +
 ",\"temperature\":" + String(temperature) +
```

```
  ",\"pressure\":" + String(pressure, 2) +
  ",\"altitude\":" + String(altitude, 2) +
  ",\"mq9\":" + sensorValue +
  "}"
);
  delay(2000);
```

РОЗДІЛ 3
ПРОГРАМНА РОЗРОБКА БОТА ТА ВЕБ-САЙТУ

Для написання коду та створення бота, ми будемо використовувати Python та відомий IDE - PyCharm, так як він створений для того щоб писати на цій мові програмування, також він дуже зручний. Будемо використовувати бібліотеку Aiogram, так як бібліотека досить часто оновлюється та є оптимізованою.

Створення telegram-бота є досить важкою роботою, для початку потрібно розібратись зі структурою файлів, для коректної та зручної розробки бота:

```
→sketch_nov9a
    ↳api
        ↳bot.py
        ↳output.json
        ↳handlers.py
        ↳keyboard.py
        ↳output.py
        ↳requests
            ↳requests_reading_create.py
            ↳requests_reading_get_all.py
            ↳requests_reading_get_last_read.py
    ↳venv
    ↳config.py
    ↳requirements
    ↳run.py
    ↳sketch_nov9a
```

3.1 Розробка Боту

Практично в кожному проєкті на мові програмуванні Python використовується віртуальне середовище(venv). І перед тим, як встановлювати фреймворк Django, Aiogram та інші бібліотеки, розглянемо детальніше про це. Віртуальне середовище потрібно для того, щоб встановлювати бібліотеки, залежності, фреймворки, пакети незалежно від глобальної системи Python.

В кожному проєкті є свої бібліотеки, залежності і т.д. , і для того, щоб не встановлювати на комп'ютер всі бібліотеки світу, які займають багато пам'яті, вони встановлюються в окремому середовищі і через одну команду можна встановити всі необхідні бібліотеки, для цього створюється текстовий файл requirements.txt через команду в консолі:

pip install -r requirements.txt

Але перед тим, створимо своє віртуальне середовище з назвою venv, для цього виконаємо певну команду в консолі:

python -m venv venv

Утворилася папка .venv щоб увійти в віртуальне середовище, виконуємо команду в консолі:

.venv/scripts/activate

Щоб впевнитись, що ми дійсно увійшли в середовище, потрібно подивитися в чат консолі, і перед консольною командою на текст, якщо з'явився текст (venv), значить ми увійшли в середовище, і можемо розпочати роботу з Ботом та Django проєктом (рис. 3.1).

Рис. 3.1. Віртуальне середовище

Після перевірки структури, створимо Telegram-бот, через інший бот - BotFather, та отримає Токен бота, для його створення та підключення до програми (рис. 3.2).

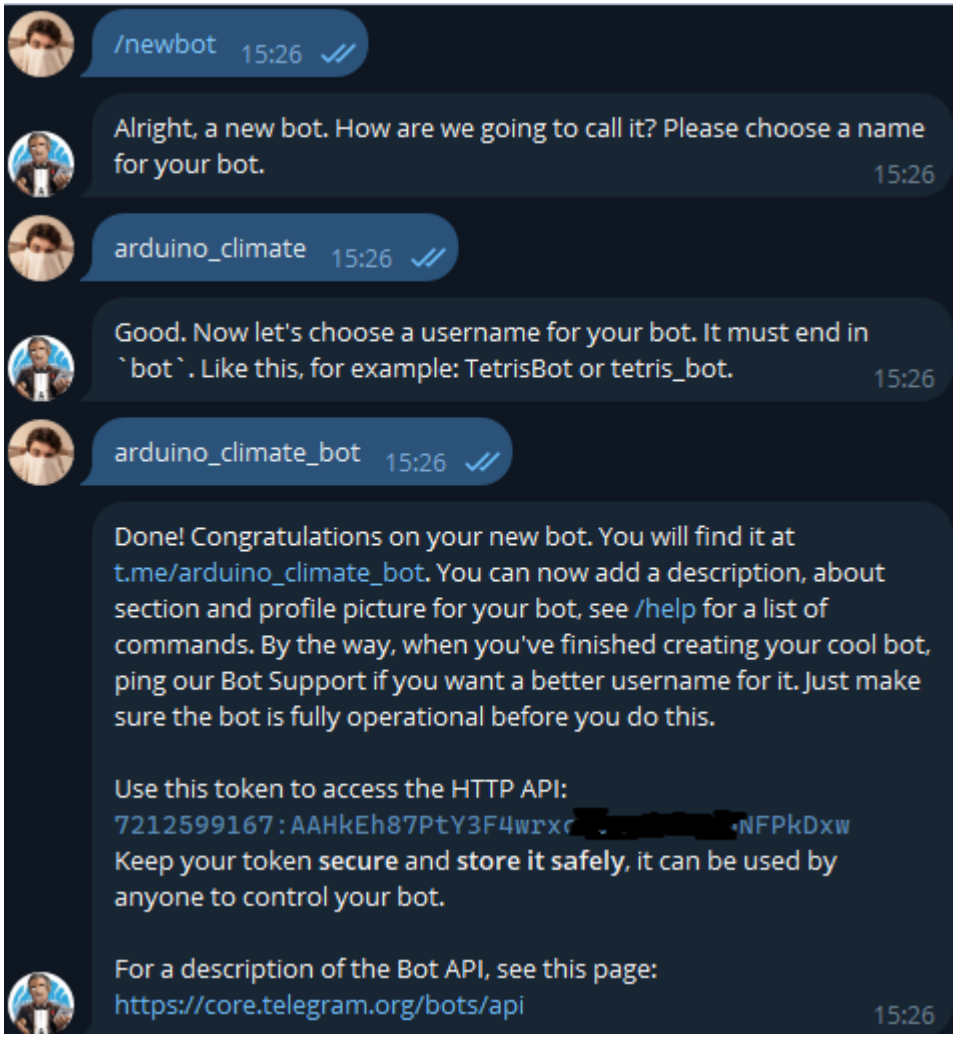


Рис. 3.2. Створення телеграм бота

Після створення бота, отримуємо Токен та прописуємо його в файл config.py для його налаштування. В файлі run.py імпортуємо раніше створений Token, router, Bot Dispatcher, так як бот потребує асинхронні методи, також імпортуємо asyncio.Створюємо змінну bot та передаємо йому Token який отримали раніше. Також змінну dp для Dispatcher, щоб обробляти вхідні оновлення, фільтрувати вхідні події і т.д. В Функція main підключаємо

імпортований router, який підключить всі обробники, які він виконує. І також в await передаємо bot який в собі містить Токен, для того, щоб бот почав працювати.

```
import asyncio
from aiogram import Bot, Dispatcher
from config import TOKEN
from api.handlers import router
import json
from datetime import datetime
bot = Bot(token=TOKEN)
dp = Dispatcher(bot=bot)
user_id_for_alert = "486940303"
```

Після створення та наповнення кодом файлу run.py приступаємо до розробки команд, на які бот буде реагувати. Перш за все потрібно імпортувати types**,** Router, для створення команди та обробників подій, та CommandStart()**,** Command для створення привітального повідомлення та подальших команд. Імпортували функції, далі потрібно створити привітальне повідомлення.

Створюємо обробник подій, як аргумент вводимо CommandStart**()**, створюємо функцію start_handler**()**. Як аргумент функції записуємо types який ми імпортували раніше та функцію Message. Та через await виводимо повідомлення в чат.

До файлу handlers**.**py повернемось пізніше, потрібно зчитати дані з датчиків, для цього створюємо файл під назвою output**.**py та пишемо код, щоб вивести дані з датчиків та перенести їх в бот. Створення json-файлу потрібно для того, щоб відсіювати інформацію яка буде йти в бот та БД, так як інформації буде багато, бо вимірювання проходить постійно, то для бази даних це буде велика нагрузка, приймати в себе велику кількість інформації в короткий час та виведення інформації з БД.

Спочатку була проблема, так як немає бібліотек в мові програмування Python на датчики, тому виводимо дані за допомогою бібліотеки “serial”, так як раніше програмували через Arduino IDE, і мікроплата вже запрограмована, потрібно тільки вивести дані.

Імпортуємо бібліотеку serial – для виведення даних з датчиків, json – для введення виведеної інформації з датчиків в файл output.json, і time – для збереження дати коли було зроблене вимірювання.

Створюємо змінну для читання з Com5 та адреси 9600.Також потрібно створити змінну для файлу, куди будуть відправлятись дані.

```
import json
import time
import serial
ser = serial.Serial('COM4', 9600)
json_filename = 'output.json'
data_records = []
```

Зчитуємо дані із серійного порту, отримуємо дані. Дані вже мають вигляд json, так як формування даних відбувається на самій програмованій Arduino платі, тому ніяких маніпуляцій та переведення даних в цей формат не потрібно робити, просто відкриваємо файл через “with open”, та записуємо дані в json-файл для подальшої роботи з ними.

```
try:
    while True:
        if ser.in_waiting > 0:
            line = ser.readline().decode('utf-8').strip()
            print("Отримано дані:", line)
            try:
                data = json.loads(line)
                data["measurement_time"] = time.strftime("%Y-%m-%d %H:%M:%S")
```

```
            data_records.append(data)
            with open(json_filename, 'w') as json_file:
                json.dump(data_records, json_file, indent=4)
        except json.JSONDecodeError:
            print("Помилка розбору JSON:", line)
except KeyboardInterrupt:
    print("Програма зупинена вручну.")
finally:
    ser.close()
```

В цьому коді і в інших майбутніх, часто використовується блок “try{}catch{}”, який дозволяє знаходити та ефективно обробляти помилки та забезпечувати стабільну роботу додатку, навіть при помилках.

Створимо команду, для моніторингу мікрокліматичних показників, для цього повернемось в файл handlers.py та створимо новий обробник подій за допомогою @router_message та раніше імпортованої функції Command, під назвою “monitor”.

Відкриваємо файл output.json, за допомогою раніше використаної команди with open, зчитуємо останні дані з файлу. Визначаємо змінні для телеграм боту в виді останніх 6 зчитаних показників: температура, вологість, тиск, висота над рівнем моря та показники датчика газу.

```
@router.message(Command(commands=["monitor"]))
async def monitor_sensors(message: types.Message):
    try:
        with open('api/output.json', "r") as json_file:
            data_records = json.load(json_file)
        # Отримуємо останній запис
        last_record = data_records[-1]
```

```
temperature = last_record['temperature']
pressure = last_record['pressure']
altitude = last_record['altitude']
humidity = last_record['humidity']
mq9 = last_record['mq9']
```

Також, створюємо запит для запису даних в БД і відправляємо звіт з актуальними даними в чат . В іншому випадку, якщо є якась помилка, ловиться помилка в чат.

```
if last_record:
    create_reading_and_alert(
        temperature,
        pressure,
        altitude,
        humidity,
        mq9
    )
    print('Дані записані в бд')
```

Формуємо звіт, табулюємо та вставляємо деякі емодзі для кращого сприйняття людиною даних.

```
response = (
 f"🕒 Час вимірювання: {last_record['measurement_time']}\n"
 f"🌡️ Температура: {temperature} °C\n"
 f"💧 Вологість: {humidity} %\n"
 f"📈 Тиск: {pressure} мм рт. ст.\n"
 f"📍 Висота: {altitude} м\n"
 f"🔥 Газ MQ9: {mq9}"
```

```
)
await message.answer(response, reply_markup=kb.read_menu)
```

В результаті створення команди, є можливість побачити результат - гарний вивід інформації та інтуїтивно зрозумілі дані (рис. 3.3).

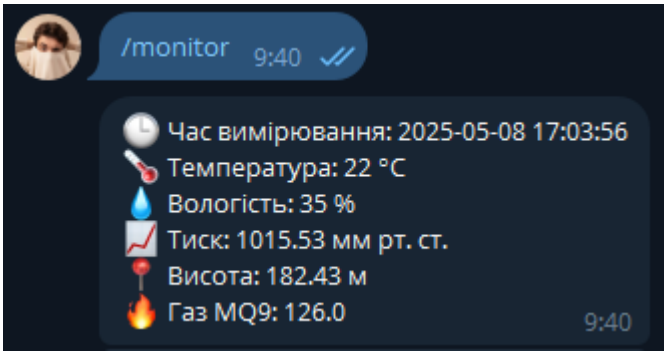


Рис. 3.3. Результат команди /monitor

В aiogram є можливість створити фонову задачу, яка буде виконуватись періодично, і є можливість її використати.Розробимо фонову команду, яка буде пересилати звіт автоматично, протягом деякого часу, так який буде записуватися в базу даних, для формування майбутніх графіків.

Для фонової команди, створюємо асинхронну функцію під назвою create_reading_every_10_minutes(), створюємо вічний цикл, щоб задача працювала постійно, відкриваємо файл output.json, в якому міститься інформація показників та вибираємо останні виміряні дані.

```
async def create_reading_every_10_minutes():
    while True:
        try:
            with open('api/output.json', 'r') as f:
                data = json.load(f)
            if data:
                latest = data[-1]
```

Створюємо змінні для показників:

```
temperature = float(latest.get("temperature",  0))
```

```
pressure = float(latest.get("pressure", 0))
altitude = float(latest.get("altitude", 0))
humidity = float(latest.get("humidity", 0))
mq9_value = float(latest.get("mq9", 0))
```

Формуємо запит для запису останніх даних в Базу даних, вписуємо показники, змінні які ми створили раніше:

```
create_reading_and_alert(
    temperature=temperature,
    pressure=pressure,
    altitude=altitude,
    humidity=humidity,
    mq9_value=mq9_value
    )
```

Перевіряємо роботу методу, чекаємо 10 хвилин і приходить наступне повідомлення (рис. 3.4):

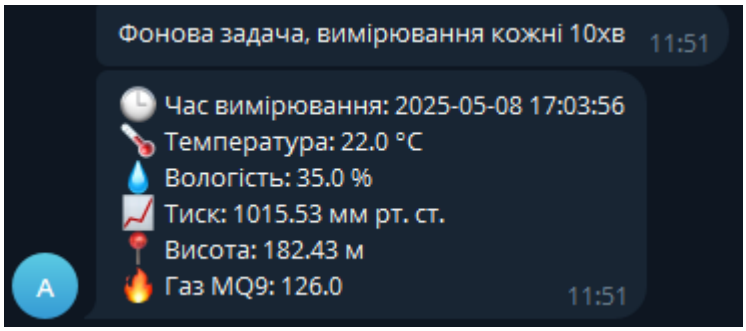


Рис. 3.4. Вивід інформації фонової задачі

Так, як одна з задач проєкту є безпека при задимленні та моніторинг інших газів, створимо фонову задачу, для отримання повідомлення від боту, про небезпеку, крім того, сформуємо ще декілька функцій, які будуть перевіряти чи занадто висока/низька температура та вологість.

Створюємо нову асинхронну функцію, яка буде періодично надсилати повідомлення про небезпеку, прописуємо змінні last_alert_time, alerted для

зберігання часу останнього повідомлення та чи було вже надіслано повідомлення про небезпечну ситуацію:

```
async def monitor_gas_sensor():
    last_alert_time = None
    alerted = False
```

Прописуємо вічний цикл, для того, щоб метод працював постійно та відкриваємо output.json для зчитування даних з датчику, вибираємо останній елемент в списку і створюємо змінну “value” для збереження показнику:

```
while True:
        try:
            with open('api/output.json', 'r') as json_file:
                data = json.load(json_file)

            if data:
                latest = data[-1]
                value = float(latest["mq9"])
```

Прописуємо логіку умовний блок if для перевірки, чи перевищує кількість газу в повітрі норму, якщо перевищує, надсилається повідомлення користувачу про певну димову небезпеку:

```
                if value > 250:
                    if not alerted or (datetime.now() - last_alert_time).total_seconds() > 10:
                        if user_id_for_alert:
                            await bot.send_message(
                                user_id_for_alert,
                                f"⚠ Увага! Виявлено дим або газ!\nЗначення датчика: {value}"
                            )
                        alerted = True
```

```
            last_alert_time = datetime.now()
        else:
            alerted = False
```

В коді присутній try{}catch{} блок для “ловлення” помилок, для того, щоб програма сама не виключалась та для знаходження де саме виникла помилка і яка саме, функція перевіряє стан повітря кожні 2 хвилини, для цього використовується функція “sleep()”:

```
    except Exception as e:
        print(f"[!] Помилка моніторингу: {e}")
    await asyncio.sleep(120)  # Перевірка кожні 2 хвилини
```

По такому самому принципу створюємо інші функції для перевірки температури та вологості: створюємо функцію, пишемо вічний цикл, відкриваємо файл, зчитуємо останні дані, перевіряємо в умовному блоці, якщо значення перевищує допустимі, то надсилаємо користувачеві (рис. 3.5).

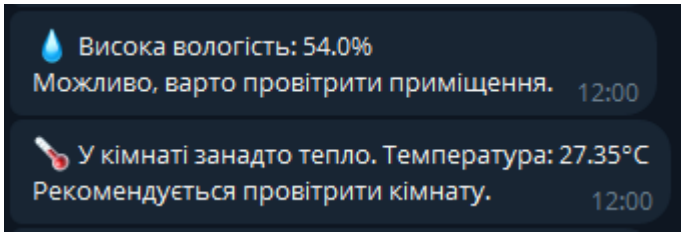


Рис. 3.5. Результати фонової функції

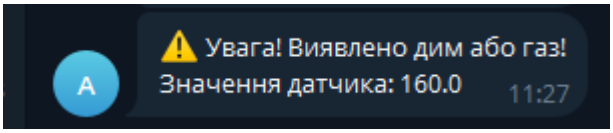


Рис. 3.6. Небезпека Газу

Якщо просто розписати функції, то вони працювати не будуть, їх потрібно почати виконувати, в Aiogram є можливість створити фонову задачу через метод asyncio.create_task(). Метод створює асинхронне завдання в події циклу та виконує його в фоновому режимі, не блокуючи виконання основного коду.

Переходимо в основний цикл main(), де виконується роздача боту, та прописуємо наші фонові функції:

```
async def main():
    dp.include_router(router)
    asyncio.create_task(create_reading_every_10_minutes())
    asyncio.create_task(monitor_gas_sensor())
    asyncio.create_task(monitor_temperature_sensor())
    asyncio.create_task(monitor_humidity_sensor())
    await dp.start_polling(bot)
```

3.2. Створення Django-проєкту сайту

Для початку роботи з фреймворком Django, потрібно встановити деякі необхідні бібліотеки. В мові програмування Python це відбувається через консольну команду pip install. Pip це система керування встановлення пакетів та бібліотек, зазвичай він встановлюється разом з мовою Python, тому pip не потрібно встановлювати окремо .

Для встановлення Django скористуємось консольною командою:

pip install Django

Також, для створення ендпоїнтів, які потрібні для передачі інформації з бази даних для сайту, встановимо Django RestFramework:

pip install djangorestframework

Створення проєкту на фреймворці Django також виконується через команди, тому скористаємось введенням команди в консоль [8]:

django-admin startproject arduino_climate .

Отже, головний додаток має наступний вигляд:

```
arduino_climate
    ├─manage.py
    └─arduino_climate
        └─settings.py
        └─urls.py
```

```
└─wsgi.py
└─__init__.py
```

Додаток містить:

settings.py: Містить всі налаштування сайту: які додаткі є в проєкті, підключення до бази даних, debug, шлях до папки з шаблонами, тощо.

urls.py: Містить в собі список шаблонів, які використовуються за певними адресами на сайті.

manage.py: Файл, який керує всім проєктом, через нього будемо запускати проєкт,створювати додатки, створювати міграції, проводити міграцію і т.д.

Для роботи з ендпоїнтами та сторінками сайту, реалізуємо це через різні додатки, створимо api_site, для роботи api, та site, для роботи з сторінками сайту, для цього використовуємо вже створений файл manage.py в консолі :

python manage.py startapp api_site

python manage.py startapp site

Щоб Django проєкт визначив додатки, та почав працювати з ними, потрібно зареєструвати їх в раніше створеному файлі settings.py. Знаходимо список додатків INSTALLED_APPS та прописуємо їх там, також пропишемо бібліотеку rest_framework для роботи з api.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'sites',
    'api_site',
```

]

3.3. Робота з API

Для роботи з API використовується вже раніше встановлений фреймворк для Django - Django Rest Framework. Rest Framework використовується для створення Restful API. Цей фреймворк дозволяє розробляти інтерфейси програмування для взаємодії між клієнтською частиною та серверною.

Основні задачі фреймворку наступні:

1) Створення Ендпоїнтів API: Передача даних в вигляді JSON.
2) Підтримка методів HTTP: PUT, DELETE, GET, POST для роботи з даними.
3) Реалізація логіки: Фільтрація, сортування даних.

Для розробки API, спочатку потрібно ініціалізувати базу даних, для цього в файлі models.py прописуємо моделі бази даних: Назву моделі, назви полів, типи даних полів.

Створюємо модель для звітів:

```
class EnvironmentReading(models.Model):
    measurement_time = models.DateTimeField(auto_now_add=True)
    temperature = models.FloatField(null=True, blank=True)
    pressure = models.FloatField(null=True, blank=True)
    altitude = models.FloatField(null=True, blank=True)
    humidity = models.FloatField(null=True, blank=True)
    mq9_value = models.FloatField(null=True, blank=True)
```

Також розробимо модель бази даних для сигналізації:

```
class Alert(models.Model):
    ALERT_TYPES = [
        ('smoke', 'Газ/Дим'),
        ('temperature_low', 'Низька температура'),
        ('temperature_high', 'Висока температура'),
        ('humidity_high', 'Висока вологість'),
```

```
        ('nothing','чисто')
    ]
    reading = models.ForeignKey(EnvironmentReading, on_delete=models.CASCADE, related_name='alerts')
    alert_type = models.CharField(max_length=30, choices=ALERT_TYPES)
    alert_message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

Щоб база даних ініціалізувалася, потрібно створити міграції та провести їх для створення бази даних зі створеними моделями, для цього виконуємо команди для ініціалізації міграції та проведення її проведення:

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

Для створення ендпоїнтів та для взаємодії з базою даних, потрібно створити Serializers, вони потрібні для перетворення об'єктів бази даних в json формат та обробку вхідних json даних для перетворення їх python-об'єкти, це потрібно при деяких запитах PUT, POST для збереження даних користувача в БД.

Створюємо файл serializers.py та прописуємо серіалайзери для наших моделей бази даних:

```
class EnvironmentReadingSerializers(serializers.ModelSerializer):
    class Meta:
        model = EnvironmentReading
        fields = '__all__'
class AlertSerializers(serializers.ModelSerializer):
    class Meta:
        model = Alert
        fields = '__all__'
```

Наступний крок для створення API - це представлення(views). Представлення потрібне для роботи з запитами користувачів та відповідей для них.

Є декілька видів запитав з якими ми будемо працювати і для яких ми будемо створювати ендпоїнти:

GET - Використовується для отримання даних від деякого ресурсу.

POST - Використовується для надсилання даних до деякого ресурсу.

DELETE - Використовується для видалення даних з деякого ресурсу.

PUT - Використовується для коригування або зміни даних з деякого ресурсу.

Отже створимо файл api_site/views.py для представлень моделі EnvironmentReading де зберігаються дані про звіти мікрокліматичних показників. Напишемо представлення для створення звіту методом POST.

Пишемо декоратор та передаємо йому запит POST, створюємо функцію, та даємо йому тіло запиту. В тілі функція створюємо умовний блок, в якому перевіряємо чи дійсно прийшов запит POST, якщо так, то серіалізуються дані та зберігаються в БД, якщо ні, то повертається помилка 400 BAD REQUEST.

```
@api_view(['POST'])
def create_environment_reading(request):
    if request.method == 'POST':
        serializer = EnvironmentReadingSerializers(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Та створимо представлення для отримання звіту методом GET.

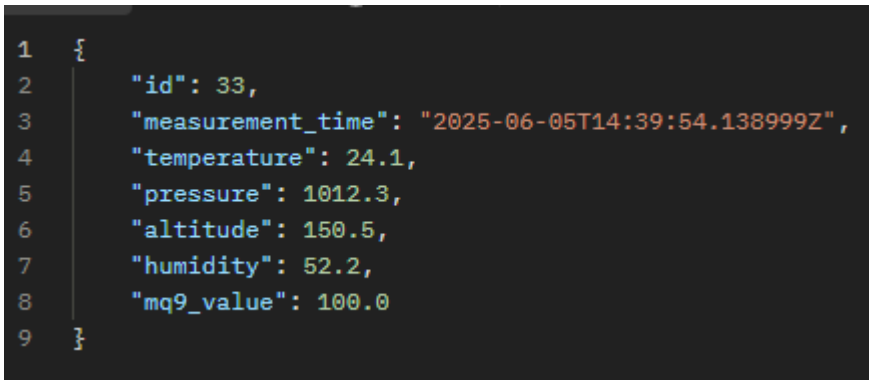
```
@api_view(['GET'])
def get_all_environment_readings(request):
```

```
    readings = EnvironmentReading.objects.all().order_by('measurement_time')
    serializer = EnvironmentReadingSerializers(readings, many=True)
    return Response(serializer.data)
```

Для того, щоб запити почали працювати, створимо файл api_site/urls.py та зареєструємо маршрути для запитів, це потрібно для того, щоб django знав, які URL-адреси повинні викликати функції, в нашому випадку це ендпоїнти:

```
urlpatterns = [
    #Reading endpoints
    path('readings/', get_all_environment_readings, name='get_all_readings'),
    path('readings/create/', create_environment_reading, name='create_reading'),
    path('readings/<int:reading_id>/', get_environment_reading_by_id, name='get_reading_by_id'),
    path('readings/<int:reading_id>/delete/', delete_environment_reading_by_id, name='delete_reading'),
    path('readings/latest/', get_latest_reading, name='latest_reading'),
    #alerts endpoints
    path('alerts/', get_all_alerts, name = 'get_all_alerts'),
    path('alerts/create/', create_alert, name='create_alert'),
    path('alerts/reading/<int:reading_id>/', get_alerts_by_id, name='get_alert_by_id'),
]
```

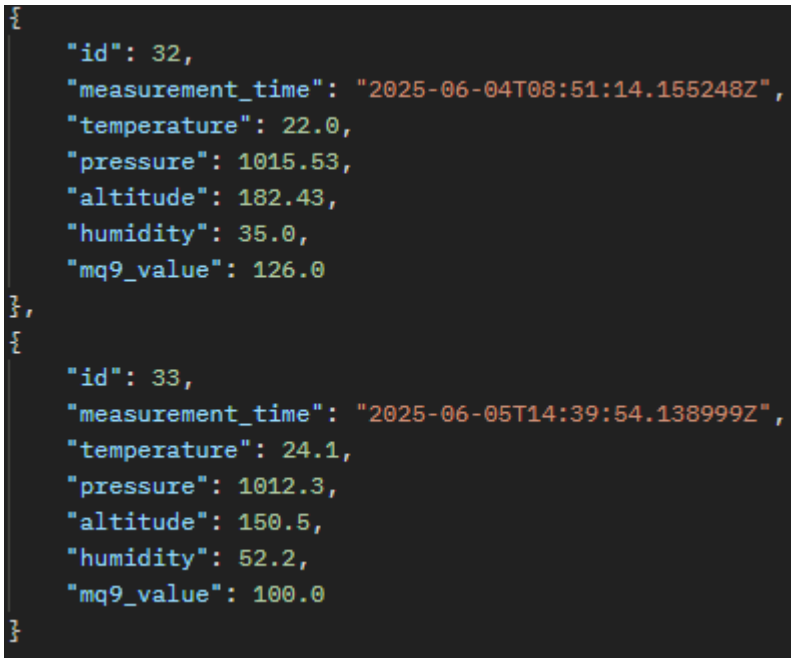
Протестуємо певні запити та перевіримо чи правильно повертаються дані та чи записуються дані в базі даних. Для чого надішлемо json-запит через програму Postman. Запит POST успішно виконався(рис.3.7).



```
{
    "id": 33,
    "measurement_time": "2025-06-05T14:39:54.138999Z",
    "temperature": 24.1,
    "pressure": 1012.3,
    "altitude": 150.5,
    "humidity": 52.2,
    "mq9_value": 100.0
}
```


Рис. 3.7. Робота POST-запиту

Перевіримо чи появились дані в базі даних через GET-запит. Дані успішно з'явилися в базі даних, та через GET-метод є можливість їх побачити(рис.3.8).



```
{
    "id": 32,
    "measurement_time": "2025-06-04T08:51:14.155248Z",
    "temperature": 22.0,
    "pressure": 1015.53,
    "altitude": 182.43,
    "humidity": 35.0,
    "mq9_value": 126.0
},
{
    "id": 33,
    "measurement_time": "2025-06-05T14:39:54.138999Z",
    "temperature": 24.1,
    "pressure": 1012.3,
    "altitude": 150.5,
    "humidity": 52.2,
    "mq9_value": 100.0
}
```


Рис. 3.8. Робота GET-запиту

Для коректної роботи URL-адресів, в головному додатку в файлі arduino_climate/urls.py розширимо список URL:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', main_view,name='main-view'),
    path('', include('sites.urls')),
```

```
    path('', include('api_site.urls')),
]
```

Це потрібно зробити, так як Django додаток, бачить адреса тільки в головному додатку, тому через функцію include(), розширюємо список URL, також робимо ці маніпуляції для майбутнього використання додатку “sites”, в якому будуть створюватись сторінки для сайту.

3.4. Створення сторінок сайту

Після створення api, перейдемо до другого створеного додатку під назвою sites. Для роботи зі шаблонами для сайту, потрібно створити папку templates, та прописати її наявність в основному додатку в файлі settings.py.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [ BASE_DIR / 'templates' ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Як і в минулому розділі, для цього додатку потрібно створити представлення(views) для шаблонів. Щоб створити графік на сайті, будемо

використовувати бібліотеку MatPlotLib. Ця бібліотека створена для візуалізації даних.

Отже, напишемо представлення з графіком та таблицею, які можна переключати на різні дні для більш точного аналізу вологості.

Для початку створюємо функцію humidity_view(), який буде отримувати в аргументі запит користувача. Також, з кнопок отримуємо запит користувача, який саме графік хоче побачити та проаналізувати, для цього створюємо деякі змінні.

```
def humidity_view(request):
    today = timezone.now().date()
    mode = request.GET.get('mode', '28days')
    graph_day_str = request.GET.get('graph_day')
    table_day_str = request.GET.get('table_day')
```

Даємо дані для графіку, залежно від того який режим вибрав користувач, для цього використовуємо минулі змінні та умовний блок. Залежно від вибору користувача, беремо з бази даних інформацію та підставляємо її для x - це день коли було вимірювання, y - середня вологість за цей день, якщо це вибраний за 1 день: x - години коли було вимірювання, y - вологість яка була під час вимірювання.

```
    elif mode == '7days':
        start = today - timedelta(days=7)
        daily_avg = (
            EnvironmentReading.objects
            .filter(measurement_time__date__gte=start)
            .annotate(day=TruncDate('measurement_time'))
            .values('day')
            .annotate(avg_humidity=Avg('humidity'))
            .order_by('day')
```

```
        )
        x = [d['day'].strftime('%Y-%m-%d') for d in daily_avg]
        y = [d['avg_humidity'] for d in daily_avg]
        title = 'Середня вологість за останні 7 днів'
        graph_day = None
```

Формуємо графік за допомогою plt, які містять наступні методи:

plt.figure - формує розміри графіку

plt.plot - передаєм x та y, можна міняти маркери та колір графіку

plt.title - встановлює заголовок для графіку

plt.xlabel,ylabel - встановлює назви для x та y осей

plt.grid - встановлення сітки на графік

```
plt.figure(figsize=(10, 5))
    plt.plot(x, y, marker='o', color='green')
    plt.title(title)
    plt.xlabel('Час' if mode == '1day' else 'Дата')
    plt.ylabel('Вологість (%)')
    plt.xticks(rotation=45)
    plt.grid(True)
    plt.tight_layout()
    buf = io.BytesIO()
    plt.savefig(buf, format='png')
    buf.seek(0)
    graph_base64 = base64.b64encode(buf.read()).decode('utf-8')
    plt.close()
```

Графік створений та є можливість зміни графіку на різні дні, чи то протягом місяця чи тижня.

Частина сайту зроблена, але ще залишається зробити таблицю з даними які були записані протягом дня.

Для формування таблиці, створюємо деякі змінні, які містять в собі можливі дні, які може вибрати користувач для аналізу, зчитуємо день який був вибраним, витягуємо дані з БД та сортуємо його з початку дня до кінця.

```
available_days = (
    EnvironmentReading.objects
    .annotate(day=TruncDate('measurement_time'))
    .values_list('day', flat=True)
    .distinct()
    .order_by('day')
)
```

Останній крок в написанні представлення, це передання інформації HTML-шаблону для рендеру всіх об'єктів які присутні на сайті(графік, таблиця, кнопки).

```
return render(request, 'humidity.html', {
'graph': graph_base64,
'mode': mode,
'graph_day': graph_day,
'available_days': available_days,
'table_day': table_day,
'readings': readings,
'avg_humidity': avg_humidity,
})
```

Представлення готове, і так як воно передає інформацію HTML-шаблону, потрібно його створити. Відкриваємо папку templates та створюємо файл temperatures.html.

Прописуємо основні теги для створення html файлу та заголовок для вкладки.

```
<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8">
  <title>Вологість — графік і щоденні дані</title>
```

Створюємо css-стилі для оформлення селекторів, зображень, форми, заголовків, навігації і т.д.

```
body {
    font-family: sans-serif;
    padding: 20px;
}
.graph-container {
    margin-bottom: 40px;
}
nav a {
    margin-right: 15px;
    text-decoration: none;
    color: #333;
    font-weight: bold;
}

h1 {
    color: #222;
}
form {
    margin-bottom: 20px;
}
```

Пишемо розділ навігації для комфортного переміщення між сторінками та розділами на сайті (рис. 3.9).

```
<nav>
  <a href="{% url 'home' %}"> Головна</a>
  <a href="{% url 'temperatures' %}"> Температура</a>
  <a href="{% url 'humidity' %}"> Вологість</a>
  <a href="{% url 'reports' %}"> Звіти</a>
</nav>
```

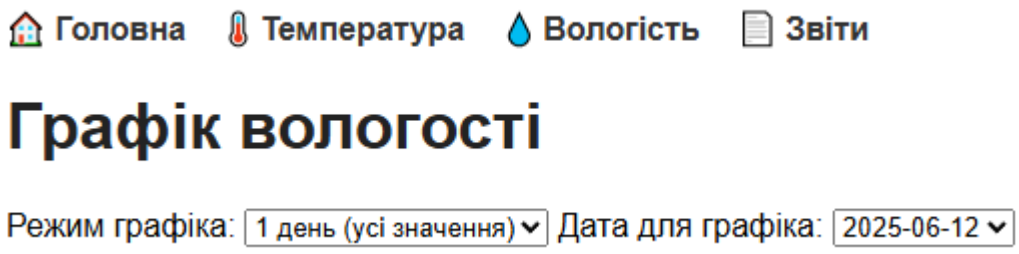


Рис. 3.9. Навігація на сайті

Створюємо кнопку для вибору користувача який типу графіку він хоче отримати, також через тег <img>, отримуємо сформований графік.

```
<form method="get">
  <label for="mode">Режим графіка:</label>
  <select name="mode" id="mode" onchange="this.form.submit()">
    <option value="1day" {% if mode == '1day' %}selected{% endif %}>1 день (усі значення)</option>
  </select>
  {% if mode == '1day' %}
    <label for="graph_day">Дата для графіка:</label>
    <select name="graph_day" id="graph_day" onchange="this.form.submit()">
      {% for day in available_days %}
```

```
            <option value="{{ day|date:'Y-m-d' }}" {% if day == graph_day
%}selected{% endif %}>
                {{ day|date:'Y-m-d' }}
            </option>
        {% endfor %}
    <div class="graph-container">
  <img src="{{ graph }}" alt="Humidity graph">
</div>
```

На рисунку (рис. 3.10) показано залежність виміряної вологості та часу.

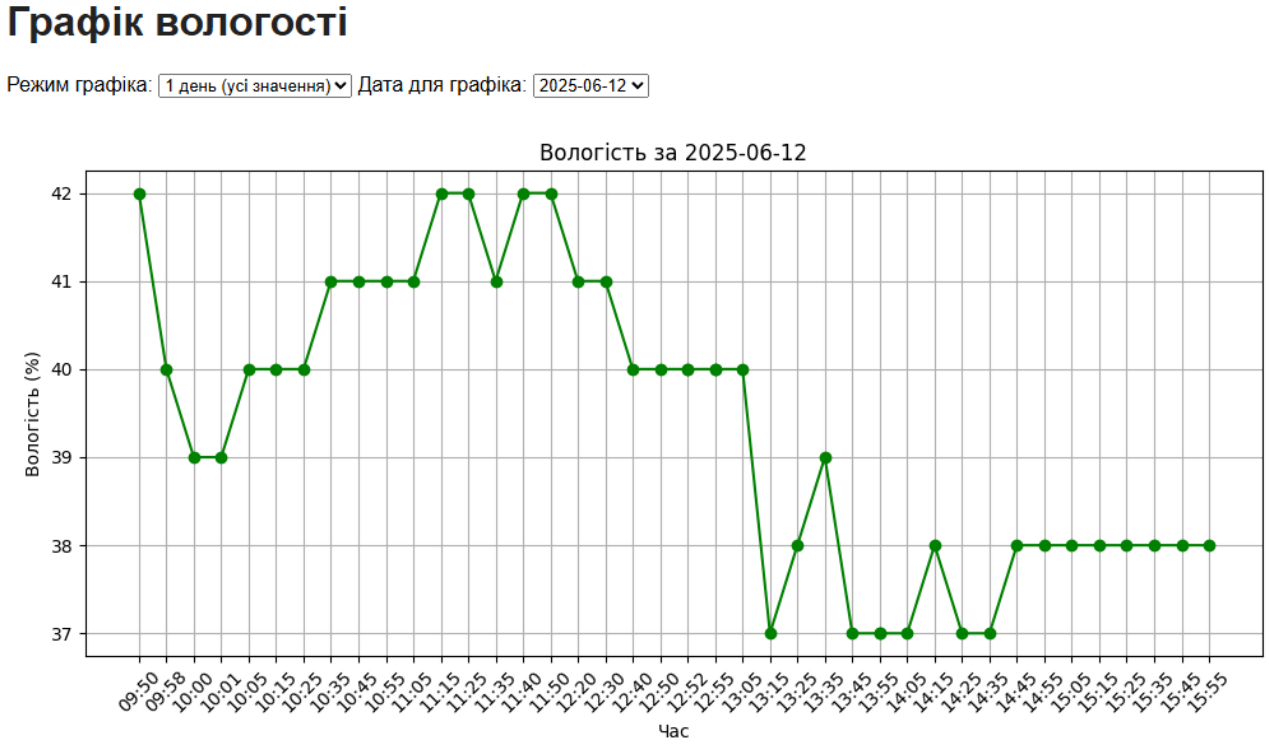


Рис. 3.10. Створений графік вологості

Для виведення історії вологості, створюємо кнопку для вибору, який день хоче вибрати користувач для ознайомлення з вологостями, і виводимо її в стилі таблиці.

```
    {% if table_day %}
        <h3>Дата: {{ table_day|date:"Y-m-d" }}</h3>
        <p>Середня вологість: {{ avg_humidity|floatformat:2 }} %</p>
```

```
    <h4>Показники за день:</h4>
    <ul>
      {% for reading in readings %}
        <li>{{ reading.measurement_time|time:"H:i" }} — {{ reading.humidity }} %</li>
      {% endfor %}
    </ul>
  {% else %}
    <p>Немає даних для обраної дати.</p>
  {% endif %}
```

Аналогічно розробляємо сторінку з температурою, підставивши тільки дані про температуру.

Створимо сторінку з звітами, для повного ознайомлення з інформацією, для цього створюємо представлення reading_view, який буде отримувати останні 100 звітів для формування історії на сайті. Звертаємось до таблиці з звітами EnvironmentReading і отримуємо дані.

```
def readings_view(request):
    readings = EnvironmentReading.objects.order_by('-measurement_time')[:100]
    return render(request, 'reports.html', {'readings': readings})
```

Щоб вивести результати, створюємо шаблон для звітів під назвою reports.html та пишемо код для виведення інформації.

Пишемо необхідні теги для формування html-файлу, та прописуємо стилі для гарного виведення інформації звітів.

```
  .reading {
        background: white;
        padding: 15px;
        margin-bottom: 10px;
```

```
        border-left: 5px solid #007BFF;
        box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    }
    .reading small {
        color: #888;
    }
    .value-row {
        margin-top: 10px;
    }
```

Створюємо кнопку для вибору дня користувачем, та виводимо інформацію про звіти користувачу.

```
{% if selected_day %}
    <h2>Дані за: {{ selected_day|date:"Y-m-d" }}</h2>
  {% for reading in readings %}
    <div class="reading">
        <strong>{{ reading.measurement_time|date:"Y-m-d H:i" }}</strong>
        <div class="value-row">
            🌡️ Температура: {{ reading.temperature }} °C<br>
            💧 Вологість: {{ reading.humidity }} %<br>
            🌫️ Тиск: {{ reading.pressure }} гПа<br>
            🏔️ Висота: {{ reading.altitude }} м<br>
            🚨 MQ-9: {{ reading.mq9_value }}
        </div>
    </div>
  {% endfor %}
```

Інформація виводиться, присутня кнопка для вибору іншого дня, та приємний вигляд дизайну (рис. 3.11).

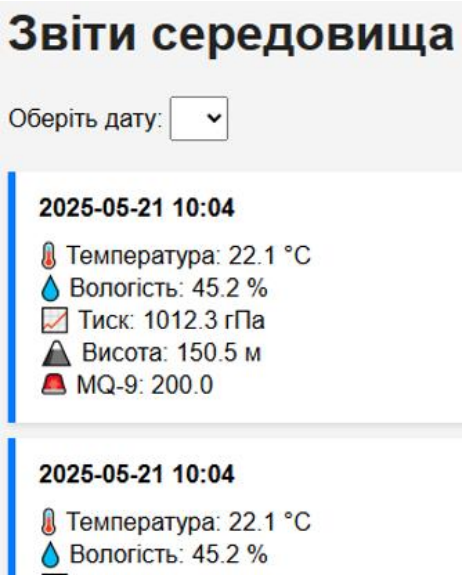


Рис. 3.11. Історія Звітів

ВИСНОВОК

В даній кваліфікаційній роботі, було розроблено систему контролю клімату та моніторингу чадного диму для моніторингу мікрокліматичних даних таких, як: температура, тиск, висота над рівнем моря, вологість та моніторинг чадного диму. Також, для цього було використано деякі пристрої та датчики, та переваги їхнього використання: Arduino Uno, DHT11, KXG1203C, BMP280 та MQ9 .

Сворений пристрій допомагає ефективно визначати мікроклімат в різних умова як і вдома так і в технічних приміщеннях. Також, увага приділяється і безпеці, система вимірює рівень чадного диму, та сигналізує це датчиком сигналізації. Для кращої сигналізаційної системи, в бот приходить повідомлення користувачу про стан повітря в приміщенні. Присутні рекомендації для користувача, які допомагають в забезпеченні кращих умов при високій/низькій температури та вологості повітря.

Використані технології такі, як: Python, C++, Django, Django Rest Framework та база даних SQLite, показали свою ефективність в розробці різних аспектів роботи: Розробка бота, API, веб додатку та бази даних.

Важливо розуміти переваги цього проєкту такі, як: доступність та відкритість, а також розширення його функціонала. Цей пристрій привабить розробників, ентузіастів та простих сімей, за рахунок своїх переваг.

Реалізація даного проекту може бути корисною для освітніх цілей, так і як для власного використання, забезпечуючи можливість швидкого та не дорогого визначення мікроклімату в домашніх умовах та безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Історія створення метеорології як науки. URL: vue.gov.ua/Метеорологія (Дата звернення: 25.04.2025).
2. Вимірювальні прилади для метеорологічних даних. URL: uk.wikipedia.org/wiki/Метеорологічна_станція (Дата звернення: 25.04.2025).
3. Характеристика плати Arduino Uno. URL: https://arduino.ua/ru/prod2610-arduino-uno-r3-ch340 (Дата звернення: 08.05.2025).
4. Характеристика датчика MQ-9. URL: https://arduino.ua/ru/prod1238-modyl-datchika-gaza-mq-9 (Дата звернення: 08.05.2025).
5. Характеристика датчика DHT-11. URL: https://arduino.ua/ru/prod185-datchik-vlajnosti-i-temperatyri-dht11 (Дата звернення: 08.05.2025).
6. Характеристика датчика BMP 280. URL: https://arduino.ua/ru/prod1758-barometr-datchik-atmosfernogo-davleniya- na-bmp280 (Дата звернення: 08.05.2025).
7. Характеристика зумера KXG1203C. URL: https://uk.rs-online.com/web/p/piezo-buzzers/7716957 (Дата звернення: 08.05.2025)
8. Django Documentation. URL: https://docs.djangoproject.com/en/5.2/ (Дата звернення: 28.05.2025).