

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

Наталія АУШЕВА

“ ” _____ 2025 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Адаптивні системи управління домашньою автоматизацією на базі ШІ та ІОТ”

Виконала: студентка 4 курсу, групи ТР-12

Імаєва Тетяна Альбертівна

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник професор каф. ЦТЕ, д.т.н., професор Сергій ОТРОХ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Рецензент професор каф. ПЗЕ, д.т.н., професор Олег БАРАБАШ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Н.контроль доцент каф. ЦТЕ Артем ГУРІН

(посада, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студентка _____

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

(підпис)

“ ” _____ 2025 р.

ЗАВДАННЯ

на дипломну роботу студентці

Імасьвій Тетяні Альбертівні

(прізвище, ім’я, по батькові)

1. Тема роботи “Адаптивні системи управління домашньою автматизацією на базі ІІІ та ІОТ ”

Науковий керівник Отрох Сергій Іванович, д.т.н., професор

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ 02 ” червня 2025 року № 1875-с

2. Термін подання студентом роботи 09.06.2025

3. Вихідні дані до роботи C++ (прошивка Arduino/PlatformIO), Python 3.x (обробка та аналіз даних), PlatformIO/Arduino IDE, PyCharm або VS Code, pandas, matplotlib, openai, python-telegram-bot, Wokwi API, CSV (часова мітка + значення показників)

4. Перелік питань, які потрібно розробити _____

1) аналіз існуючих ІОТ-систем моніторингу довкілля

2) визначення функціональних та нефункціональних вимог до інтелектуальної платформи

3) обґрунтування вибору технологій та інструментів

4) проектування загальної архітектури системи і структури даних

5) розробка та інтеграція основних модулів: збір та логування, попередня обробка і побудова графіків, інтелектуальний аналіз через ChatGPT, механізм сповіщень у Telegram

6) тестування й оцінка ефективної системи

5. Орієнтований перелік ілюстративного матеріалу схема архітектури проєкту, ER-діаграма класів, діаграма прецедентів (ролей), діаграма послідовностей,скріншоти роботи користувачів з веб-застосунком.

6. Дата видачі завдання 13.09.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	10.04.2025	виконано
2.	Аналіз методів та засобів розв’язання задачі	17.04.2025-18.04.2025	виконано
3.	Розробка архітектури та загальної структури системи	19.04.2025-22.04.2025	виконано
4.	Розробка окремих підсистем	22.04.2025-28.04.2025	виконано
5.	Програмна реалізація системи	29.04.2025-07.05.2025	виконано
6.	Оформлення пояснювальної записки	08.05.2025-12.05.2025	виконано
7.	Захист програмного забезпечення	12.05.2025-16.05.2025	виконано
8.	Передзахист	26.05.2025-28.05.2025	виконано
9.	Захист	16.06.2025-21.06.2025	виконано

Студентка

(підпис)

Тетяна ІМАЄВА

(прізвище та ініціали)

Керівник

(підпис)

Сергій ОТРОХ

(прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 74 сторінках, містить 7 рисунків, 1 таблицю, 1 додаток, 20 джерел.

Мета роботи – розробка адаптивної системи управління домашньою автоматизацією на базі Arduino з емуляцією Wokwi, збором та обробкою даних у Python (pyserial, pandas, matplotlib), аналізом трендів через OpenAI API та керуванням пристроями через Telegram-бота.

Методи та засоби: Arduino і емулятор Wokwi API Platform IDE; датчики DHT22, MQ-2, PMS5003; протоколи MQTT/HTTP; мова Python із бібліотеками pyserial, pandas, matplotlib, requests; інтеграція з OpenAI API; бібліотека python-telegram-bot; збереження телеметрії в CSV.

Результат – реалізований прототип адаптивної системи домашньої автоматизації з інтерфейсом у вигляді Telegram-бота.

Ключові слова: Інтернет речей, штучний інтелект, Arduino, MQTT, OpenAI API, Telegram-бот, емуляція Wokwi.

ANNOTATION

The diploma thesis is presented on 74 pages, with 7 illustrations, 1 table, 1 appendice, and 20 sources.

The aim of the work is the development of an adaptive home automation control system based on an Arduino sensor node (emulated via Wokwi), real-time data processing in Python (pyserial, pandas, matplotlib), trend analysis using the OpenAI API, and device management through a Telegram bot.

Methods and tools: Arduino and the Wokwi API Platform IDE; DHT22, MQ-2, and PMS5003 sensors; MQTT/HTTP protocols; Python with the pyserial, pandas, matplotlib, and requests libraries; integration with the OpenAI API; the python-telegram-bot library; and CSV for telemetry storage.

Result – a working prototype of an adaptive home automation system with a Telegram bot interface.

Keywords: Internet of Things, artificial intelligence, Arduino, MQTT, OpenAI API, Telegram bot, Wokwi emulator.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ	7
ВСТУП.....	9
1 АНАЛІЗ ЛІТЕРАТУРИ ТА ІСНУЮЧИХ РІШЕНЬ У СФЕРІ ДОМАШНЬОЇ АВТОМАТИЗАЦІЇ.....	11
1.1 Сучасні IoT-платформи для «розумного дому».....	13
1.2 Застосування методів штучного інтелекту для адаптивного керування.....	15
1.3 Інтеграція чат-ботів у системи моніторингу та управління	17
2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ АДАПТИВНОЇ СИСТЕМИ.....	20
2.1 Апаратна платформа та сенсорний вузол	21
2.1.1 Переваги емуляторної платформи.....	26
2.1.2 Віртуальні датчики в сенсорному вузлі.....	27
2.1.3 Конфігурація емулятора та віртуальних зв'язків	28
2.1.4 Калібрування та точність вимірювань в віртуальному середовищі	29
2.1.5 Інтеграція з програмною архітектурою	30
2.1.6 Підсумки емуляції сенсорного вузла	30
2.2 Комунікаційний шар і мережеві протоколи	31
2.3 Формат і сховище даних	32
2.4 Архітектура програмних модулів.....	33
2.5 Взаємодія компонентів та інформаційні потоки	36
3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	38
3.1 Модуль зчитування і попередньої обробки даних із датчиків	39
3.1.1 Архітектура та налаштування модулів	40
3.1.2 Валідація отриманих даних	41
3.1.3 Фільтрація і згладжування даних	42
3.1.4 Формування єдиного повідомлення для наступних етапів	42
3.1.5 Асинхронна обробка і відновлення після збоїв	43

3.1.6	Логування модуля та спостереження за здоров'ям.....	43
3.2	Зберігання та публікації даних	44
3.3	Інтеграція з OpenAI API для аналізу трендів і генерації рекомендацій	45
3.4	Розробка чат-бота Telegram для відображення графіків і взаємодії з користувачем	47
4	ІНТЕГРАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ.....	51
4.1	Випробування та перевірка відповідності вимогам	54
4.2	Проектування та план реалізації простого базового моніторингу.....	56
5	ВЗАЄМОДІЯ КОРИСТУВАЧА З TELEGRAM-БОТОМ	61
5.1	Інструкції з встановлення чат-бота та вимоги до системи.....	64
5.2	Функціональне тестування чат-бота	65
	ВИСНОВКИ.....	69
	СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	70
	ДОДАТОК А.....	72

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IoT (Internet of Things) – концепція мережі фізичних пристроїв (сенсори, виконавчі механізми, контролери), підключених до Інтернету й взаємодіючих між собою для збору, обміну та аналізу даних у реальному часі.

ШІ (штучний інтелект) – галузь комп'ютерних наук, що вивчає методи побудови систем, здатних імітувати людський розум: обробляти природну мову, розпізнавати образи, прогнозувати події; у дипломній роботі застосовано OpenAI API для перетворення числових трендів на зрозумілі рекомендації.

Arduino – відкрита апаратна платформа на основі мікроконтролера, призначена для швидкого прототипування електронних пристроїв; у проєкті використано як реальний вузол, так і його емуляцію в середовищі Wokwi.

Wokwi API Platform IDE (емуляція Arduino) – віртуальне середовище, яке точно моделює апаратні підключення сенсорів (DHT22, MQ-2, PMS5003) й генерує дані з CRC-контролем, дозволяючи тестувати алгоритми збору та обробки без фізичного “заліза”.

MQTT (Message Queuing Telemetry Transport) – легковаговий протокол із моделлю “publish/subscribe” для передачі невеликих повідомлень (телеметрії) з сенсорів; забезпечує мінімальне мережеве навантаження та гарантовану доставку за допомогою QoS.

HTTP (Hypertext Transfer Protocol) – протокол прикладного рівня та архітектурний стиль для обміну даними через стандартизовані методи (GET, POST тощо); у системі використано для пакетної передачі CSV-файлів із архівними даними.

CSV (Comma-Separated Values) – простий текстовий формат для зберігання табличних даних, у дипломній роботі застосовується для логування телеметрії з датчиків із ротацією й архівацією файлів.

OpenAI API – хмарний сервіс для роботи з великими мовними моделями (GPT), що приймає природномовні промпти, аналізує JSON-відповіді й повертає тексти рекомендацій із оцінкою впевненості.

Telegram-бот – інтерфейс користувача в месенджері Telegram, реалізований за допомогою бібліотеки `python-telegram-bot`; приймає команди (`/status`, `/graph`, `/recommend`), надсилає графіки, сповіщення про аномалії та адаптивні поради.

DHT22 – цифровий сенсор температури та вологості з точністю $\pm 0,5\text{ }^{\circ}\text{C}$ і $\pm 2\text{ \% RH}$; інтервал оновлення – раз на 2 с, однопровідний інтерфейс спрощує підключення до Arduino.

MQ-2 – універсальний газовий датчик для вимірювання чадного газу (CO), летких органічних сполук та диму; має аналоговий вихід і потребує прогріву перед роботою для стабілізації показань.

PMS5003 – сенсор концентрації твердих частинок (PM1.0, PM2.5, PM10) із CRC-контролем даних; в емуляторі Wokwi налаштовується через інтерфейс для тестових сценаріїв зміни значень.

ВСТУП

У сучасному світі домашня автоматизація перетворюється з екзотичної опції на необхідний атрибут комфорту та безпеки. Під впливом стрімкого розвитку Інтернету речей (IoT) та методів штучного інтелекту (ШІ) з'являються нові можливості для створення гнучких, адаптивних систем управління побутовими приладами й інженерними мережами. Інтеграція сенсорів температури, вологості, якості повітря, димових і пилових детекторів дозволяє не лише автоматизувати прості сценарії «увімкнути–вимкнути», а й реалізувати інтелектуальний аналіз стану навколишнього середовища та коригувати налаштування у режимі реального часу.

Попит на «розумні» рішення обумовлений чотирма основними факторами. По-перше, зростання вартості енергоресурсів і необхідність їх економії стимулюють пошук систем, які оптимізують роботу опалення, кондиціонування та освітлення. По-друге, питання безпеки житла і здоров'я людей актуалізують використання датчиків диму, чадного газу та інших небезпечних речовин із можливістю миттєвого оповіщення власника. По-третє, підвищення рівня технічної грамотності користувачів та доступність ІОТ-пристроїв спрощують впровадження “smart home” незалежно від масштабу та бюджету. Нарешті, можливості ШІ дають змогу передбачати поведінку систем і людей, створюючи зони з оптимальними умовами проживання.

Метою даної роботи є розробка адаптивної системи управління домашньою автоматизацією, що поєднує:

- апаратну платформу на базі Arduino із набором датчиків — температури, вологості, диму та пилу;
- інформаційний шар з використанням мережевих протоколів MQTT/HTTP для обміну даними між сенсорами, сервером і клієнтом [1];
- програмну обробку даних із застосуванням Python та API OpenAI для інтелектуального аналізу трендів та генерації рекомендацій [2];

- інтерфейс взаємодії у вигляді чат-бота (Telegram), який дозволяє у зручний спосіб отримувати поточні показники, переглядати графіки та запитувати висновки від ШІ.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

Провести аналіз існуючих IoT-рішень та платформ для домашньої автоматизації, визначити сильні й слабкі сторони різних підходів:

- розробити загальну архітектуру системи, яка забезпечує надійну передачу та зберігання даних від датчиків;
- реалізувати програмні модулі для зчитування інформації з апаратної платформи, передачі її на сервер і збереження в CSV-файлі;
- інтегрувати модуль взаємодії з OpenAI API для аналізу історичних даних і формування адаптивних керуючих команд [3];
- створити чат-бота для зручної взаємодії користувача із системою та виведення результатів у вигляді текстових повідомлень і графіків;
- Провести комплексне тестування, інтеграцію компонентів і оптимізацію продуктивності системи.

Наукова новизна дослідження полягає у застосуванні гібридного підходу: поєднанні простого, недорогого апаратного забезпечення з потужними можливостями сучасних мов програмування та ШІ-сервісів. Практична значущість роботи полягає у створенні прототипу системи, яка може бути легко масштабована та модернізована під конкретні потреби користувача або архітектуру будівлі.

1 АНАЛІЗ ЛІТЕРАТУРИ ТА ІСНУЮЧИХ РІШЕНЬ У СФЕРІ ДОМАШНЬОЇ АВТОМАТИЗАЦІЇ

Упродовж останнього десятиліття домашня автоматизація перетворилася з нішевої технології на масову сферу застосувань, причому її розвиток визначають одразу кілька ключових напрямів. По-перше, зростання доступності апаратних рішень на базі мікроконтролерів і одночасне зниження вартості сенсорів стимулювали появу дешевих платформ, які можна адаптувати під конкретні завдання. По-друге, стандарти відкритого програмного забезпечення та інтерфейсів зв'язку зміцнилися в якості де-факто галузевих рішень, спростивши інтеграцію різних пристроїв у єдину систему. Нарешті, по-третє, розвиток хмарних обчислень і серіалізація API забезпечили можливість швидкої обробки та зберігання великих обсягів даних, що виникають при постійному моніторингу умов у будинку.

У науковій літературі доволі чітко простежується розподіл підходів між централізованими та децентралізованими архітектурами. Централізований підхід передбачає наявність єдиного «мозку» — сервера чи одноплатного комп'ютера (наприклад, Raspberry Pi), який контролює всі вузли та виконавчі механізми. Така модель відзначається простотою налаштування й очевидністю потоків даних, проте створює єдину точку відмови і має обмеження на масштабування. Децентралізовані системи, навпаки, спираються на брокери повідомлень або шини даних, що дозволяє вузлам безпосередньо обмінюватися сенсорними показниками та командами без жорсткої ієрархії. Цей підхід підвищує гнучкість і резервування, але водночас ускладнює синхронізацію та може спричиняти затримки при піковому навантаженні.

Окрему гілку досліджень присвячено питанням забезпечення безпеки і конфіденційності. Оскільки компоненти домашньої автоматизації зазвичай підключені до Інтернету, вони стають потенційною точкою входу для злоумисників. Найбільш поширеними ризиками вважаються несанкціоноване прослуховування каналів комунікації й атаки типу «людина-по-серверу». У зв'язку

з цим у профільних працях рекомендують використовувати протоколи з шифруванням на транспортному рівні (TLS/SSL), а також аутентифікацію вузлів із допомогою сертифікатів й API-ключів. Окрім перенесення обробки даних на захищені хмарні інфраструктури, дослідження звертають увагу на важливість регулярного оновлення програмного забезпечення в сенсорних вузлах та контролерах для усунення вразливостей.

Користувачські інтерфейси домашніх систем еволюціонували від простих веб-консоль до мобільних додатків та голосових асистентів. У перших реалізаціях оператор умикав чи вимикав пристрої через локальні веб-сторінки, але з появою сучасних фреймворків і мобільних SDK з'явилася можливість створювати інтуїтивні додатки з підтримкою push-повідомлень, геолокаційних опцій і віддаленого доступу. Деякі дослідження вказують на підвищену задоволеність користувачів, які можуть отримувати сповіщення про критичні події (протікання води, підвищена концентрація газу) миттєво й незалежно від місцезнаходження.

У практичних впровадженнях домашньої автоматизації чималу роль відіграють питання енергоефективності. Метою є не лише забезпечити комфорт, а й мінімізувати витрати ресурсів. Відтак у літературі розглядають різні алгоритми адаптивного регулювання режимів опалення та освітлення залежно від циклів використання приміщень. Дослідження демонструють, що навіть базові правила, реалізовані на основі порогових значень температури чи рівня освітленості, здатні зекономити до 15–20 % енергії без втрати комфорту.

При цьому критичною умовою ефективності є зворотний зв'язок: система повинна відслідковувати результат прийнятих рішень і коригувати правила у разі відхилень. Тому в сучасних рішеннях передбачаються механізми логування всіх подій і побудови звітів. Звіти дозволяють власнику аналізувати ефективність режимів, отримувати графіки споживання та приймати рішення про подальші налаштування. У деяких випадках передбачена й програма оновлення логіки на основі зміни моделей поведінки користувачів чи нових умов експлуатації (наприклад, сезонна корекція порогів температури).

Узагальнюючи, огляд літератури та існуючих рішень демонструє, що успіх домашньої автоматизації залежить від вдалого поєднання надійних каналів зв'язку,

безпечних методів обміну даними, зручних інтерфейсів і можливості гнучкого налаштування сценаріїв.

1.1 Сучасні IoT-платформи для «розумного дому»

Упродовж останнього десятиліття сфера «розумного дому» суттєво розширилася завдяки появі недорогих платформ на основі мікроконтролерів і відкритого програмного забезпечення. Сьогодні користувачі можуть вибирати серед десятків ІОТ-фреймворків, кожен із яких по-своєму вирішує завдання збору, передачі, обробки та візуалізації даних від сенсорів і виконавчих пристроїв. Відкриті рішення стали особливо привабливими в науковому середовищі: вони дозволяють швидко прототипувати і кастомізувати систему, долучаючи нові датчики, алгоритми та сервіси.

Однією з найпоширеніших серед DIY-ентузіастів і дослідників є Home Assistant. Написаний на Python, він підтримує тисячі інтеграцій — від MQTT-датчиків до Bluetooth- і ZigBee-пристроїв. Архітектура побудована навколо ядра, яке виконує планувальник автоматизацій і веб-сервер, а плагіни завантажуються як компоненти при старті. Налаштовувати сценарії можна як через YAML-конфігурації, так і за допомогою вбудованого візуального редактора, що значно спрощує роботу з користувачем. Крім того, підтримка Docker дає змогу розгорнути Home Assistant на будь-якому сервері всього за кілька хвилин [1], [4].

Альтернативою є OpenHAB — рішення на базі Java, орієнтоване на максимальну кросплатформність. Воно використовує модуль «bindings» для підключення різних протоколів і пристроїв, а власна rule-engine дозволяє задавати сценарії як DSL-скрипти, так і через інтерпретовані мови (JavaScript, Python тощо). OpenHAB відзначається стабільністю на великих установках і багатим інструментарієм для інтеграції з корпоративними системами, проте потребує трохи більше часу на початкове налаштування [6].

Для легковагових установок популярний Domoticz — платформа з мінімальними залежностями, що чудово працює навіть на малопотужних

одноплатних комп'ютерах. Вона споживає вкрай мало ресурсів і підтримує як MQTT, так і традиційні 433 МГц-пристрої через USB-донгли. Інтерфейс Domoticz дозволяє відображати графіки та керувати пристроями через веб-сторінку, але має обмежені можливості для розширеного сценарного програмування [3], [5].

Універсальним інструментом багатьох проєктувальників є Node-RED — flow-based середовище з візуальним конструктором потоків даних. Воно не є самостійною «розумною» платформою, проте здатне виступати проміжним шаром між сенсорними вузлами, базами даних і зовнішніми API. Використовуючи вузли для MQTT, HTTP, WebSocket і власні JavaScript-функції, Node-RED легко інтегрується з Home Assistant, OpenHAB або будь-яким іншим брокером повідомлень, надаючи гнучкість у реалізації складної логіки [1], [2], [4], [5].

Вимоги до безпеки і надійності дедалі жорсткішають, тому сучасні платформи підтримують TLS-шифрування для MQTT і HTTPS-канали для REST-інтерфейсів. Деякі сервіси (наприклад, Home Assistant Cloud) пропонують «безпечний тунель» без налаштування роутера, а в OpenHAB доступні модулі для двофакторної аутентифікації. Регулярні оновлення й активні спільноти користувачів допомагають своєчасно закривати вразливості та обмінюватися досвідом [1], [2], [5].

У корпоративному сегменті починають використовувати платформу ThingsBoard, що пропонує масштабовані черги повідомлень, адаптивні дашборди та rule-engine із підтримкою складних подій. Для великих обсягів телеметрії часто застосовують TimescaleDB або Apache Cassandra як бекенд, тоді як самі платформи розгортаються у контейнерах Kubernetes для забезпечення високої доступності. Подібні рішення мають вищий поріг входу, однак відповідають вимогам з безпеки, моніторингу продуктивності та аудиту [6].

Не можна обійти увагою й хмарні сервіси AWS IoT Core, Azure IoT Hub та Google Cloud IoT [7]. Вони надають повний стек сервісів: від пристроїв із сертифікацією до аналітики в реальному часі та машинного навчання на базі серверлес-функцій. Хмарна архітектура гарантує автоматичне масштабування та інтеграцію з іншими корпоративними ресурсами, але вимагає інтернет-з'єднання й додаткових витрат. Багато гібридних рішень використовують локальний MQTT-

брокер із дублюванням даних у хмару для аналізу й довготривалого зберігання [8],[9],[10].

Отже, сучасні ІОТ-платформи для «розумного дому» пропонують широкий спектр можливостей: від легких DIY-рішень до корпоративних і хмарних інфраструктур. Вибір конкретного фреймворку залежить від завдань, вимог до надійності, ресурсів обладнання та рівня технічної підготовки розробників. Для академічних досліджень і прототипування найчастіше обирають Home Assistant або OpenHAB через їхню гнучкість, активні спільноти та багатий набір інтеграцій [11], [12].

1.2 Застосування методів штучного інтелекту для адаптивного керування

Застосування штучного інтелекту в системах домашньої автоматизації відкриває широкі можливості для створення адаптивних механізмів керування, які здатні не лише виконувати заздалегідь задані сценарії, а й самостійно прогнозувати зміни умов середовища та оптимізувати режим роботи пристроїв. Головна ідея полягає в тому, щоб на основі історичних даних і поточних показників сенсорів формувати керуючі дії, що забезпечують максимальний комфорт і мінімальне енергоспоживання.

Найпоширенішим підходом є використання моделей часових рядів для прогнозування параметрів мікроклімату приміщення. Наприклад, реалізація LSTM-нейромереж дає змогу передбачити зміну температури та вологості на горизонті 6–12 годин із похибкою, що не перевищує 0,5 °C і 3 % відповідно. Таке прогнозування дозволяє автоматично коригувати роботу опалення, кондиціонерів або осушувачів повітря ще до настання критичних значень, завдяки чому зростає ефективність використання енергії і підвищується стабільність клімату в кімнаті [12].

Для адаптивного керування освітленням та затримками ввімкнення/вимкнення приладів використовують методи кластеризації та

класифікації. Наприклад, алгоритм K-means групує події ввімкнення світла за часом і тривалістю роботи, що дозволяє виявити типовий розклад мешканців. На підставі отриманих кластерів система будує прогностичні правила: якщо поточний день і година відповідають одній із виділених груп, світло вмикається автоматично за півхвилини до звичного часу, а вимикається — після фіксованої затримки. Такі рішення знижують кількість зайвих ручних дій і дають відчуття «навченості» системи звичкам користувача.

Окрему увагу приділено виявленню аномалій у показниках безпеки: концентрації диму, пилу чи летких речовин. Для цього застосовують алгоритми One-Class SVM або Isolation Forest, які навчаються на «здорових» даних і спрацьовують, коли поточні показники суттєво відхиляються від норми. У разі виявлення аномалії система негайно надсилає сповіщення власнику та може автоматично вмикати вентиляцію або сигналізацію. Використання таких алгоритмів гарантує швидку реакцію на потенційні загрози навіть без чіткого порогового значення в налаштуваннях [13].

Для більш гнучких і складних сценаріїв застосовують методи підкріпленого навчання (reinforcement learning). У рамках адаптивного керування кліматом агент (система) взаємодіє з середовищем (приміщенням) через набір можливих дій (увімкнути опалення, зменшити подачу повітря, тощо) і отримує винагороду, що залежить від комфортності (на основі зворотного зв'язку від користувача або внутрішньої метрики) та рівня енергоспоживання. Прості Q-learning алгоритми дозволяють навчитися оптимальному набору дій для різних умов на основі ітеративного проб і помилок. Проте для реальних сценаріїв частіше використовують Deep Q-Networks або Policy Gradient, залучаючи нейромережі для оцінки складних станів середовища [14].

Ще одним перспективним напрямом є використання нечіткої логіки (fuzzy logic). На її основі будують системи, які замість чітких порогів можуть працювати з «розмитими» правилами: «якщо температура трошки вища за комфортну, зменшити обігрів помірно», «якщо вологість значно нижча, то включити осушувач на максимальній швидкості». Нечіткі контролери поєднують достоїнства експертних систем і стандартних П-І-Д-регуляторів, спрощуючи опис доменної

логіки та забезпечуючи природнішу поведінку при близьких до граничних умовах [15].

Інтеграція з хмарними ML-сервісами, зокрема OpenAI API, розширює можливості інтерпретації даних і формування рекомендацій у природній мові. Наприклад, за допомогою GPT-моделі можна генерувати пояснення змін клімату в приміщенні та пропозиції для користувача: «Упродовж останніх трьох годин температура регулярно зростала на 0,7 °C через підвищену сонячну активність — рекомендується знизити налаштування обігрівача або заздалегідь провітрити кімнату». Такий підхід підвищує довіру та полегшує сприйняття автоматичних рішень [10].

При впровадженні ШІ-методів важливо врахувати обмежені ресурси сенсорних вузлів. Тому використовують й гібридні архітектури: на самому пристрої виконуються тільки легкі обчислення (агрегація, виявлення простих подій), а «важкі» задачі (навчання мережі, складне прогнозування) делегуються серверу або хмарі. Для зменшення затримок критичні модулі можуть бути винесені на локальний edge-сервер (Raspberry Pi, NUC), тоді як історичні дані зберігаються у хмарі для довготривалої аналітики.

Використання методів штучного інтелекту в домашній автоматизації дозволяє створити по-справжньому адаптивні системи, які навчаються на поведінці мешканців, прогнозують зміни умов і приймають оптимальні рішення в автоматичному режимі. Такий підхід вже сьогодні доводить свою ефективність у зниженні енергоспоживання, підвищенні комфорту та забезпеченні безпеки, а в перспективі — наближає «розумний дім» до рівня інтелектуального помічника, який дбає про мешканців без їхньої безперервної уваги.

1.3 Інтеграція чат-ботів у системи моніторингу та управління

У сучасних IoT-системах домашньої автоматизації чат-боти виступають не просто як зручний інтерфейс комунікації, а як повноцінний шар логіки, що забезпечує динамічну взаємодію між користувачем і «розумним» середовищем.

Основна ідея полягає в тому, щоб чат-бот виконував роль універсального консолі оператора: приймати запити, передавати їх у внутрішню систему контролю, отримувати дані з сенсорів та у зворотному зв'язку відповідати людиною зрозумілими повідомленнями.

По-перше, інтеграція починається з вибору каналу комунікації. Найчастіше застосовуються Telegram і Slack: вони надають простий API, HTTPS-канали для веб-хуків та готові SDK для більшості мов програмування. Для реалізації необхідно створити «бот-акаунт», отримати токен доступу та налаштувати зворотній виклик (webhook) на адресу сервера або мікросервісу, який обробляє вхідні повідомлення. Кожен запит користувача у вигляді тексту, командної шкали або кнопок інлайн-меню перетворюється на подію, що передається до головного сервісу обробки.

По-друге, у системі моніторингу чат-бот відповідальний за маршрутизацію інформації: він підписується на внутрішні події IoT-шини (MQTT-топіки або події REST API) і, за потреби, публікує дані в особистий чат користувача. Наприклад, повідомлення про перевищення порогового рівня диму або раптове різке зниження вологості надсилається миттєво, без запиту від користувача. Цей «push-механізм» забезпечує оперативне інформування й дозволяє реагувати на позаштатні ситуації у режимі реального часу [16].

По-третє, чат-боти можуть генерувати візуальний контент: графіки зміни температури, вологості та інших показників. Для цього на сервері реалізуються модулі побудови діаграм (наприклад, на базі matplotlib або Plotly), які зі збережених даних формують зображення та надсилають їх у чат у вигляді файлу або зображення. Такий підхід дає змогу користувачу швидко оцінити динаміку змін без необхідності переходити в окремий веб-інтерфейс.

По-четверте, інтеграція керування передбачає реалізацію команд управління через бот. Користувач може надсилати кнопки типу «Увімкнути обігрівач», «Перевірити рівень пилу» або вводити текстові команди `'heater on 22'` для встановлення бажаної температури. Сервер обробляє ці запити, переводить їх у відповідні API-виклики контролера (наприклад, публікацію MQTT-повідомлення в топик `'home/heater/set'`) і повертає підтвердження або помилку. Завдяки інтелектуальній обробці тексту (наприклад, через прості NLP-бібліотеки або навіть

ChatGPT-адаптер), бот може розпізнавати більш складні запити на кшталт «Постав температуру на рівні, як вчора о 9 вечора» [5].

По-п'яте, забезпечення безпеки та автентифікації під час інтеграції чат-бота потребує захисту токенів API та контролю доступу. Рекомендується використовувати механізми двофакторної автентифікації (2FA) для розмежування прав різних користувачів: наприклад, кілька сімейних акаунтів можуть мати різні рівні доступу до критичних команд (від перегляду даних до зміни налаштувань). Сервер бота перевіряє ідентифікатори відправника та може виводити в чат лише ті елементи інтерфейсу, які дозволені конкретній ролі (наприклад, гість бачить лише статус, а адміністратор — керує сценаріями).

По-шосте, автоматизація через чат-боти охоплює використання планувальників і сценаріїв. Бот може відправляти щоденні або щотижневі звіти про стан системи, режим роботи опалення чи кількість спрацьовувань детекторів. Планувальник на сервері формує текстовий звіт із коротким викладом основних показників і відсилає його у фіксований час. Такий механізм дозволяє регулярно контролювати ефективність і своєчасно виявляти тренди.

По-сьоме, інтеграція з ChatGPT або локальними NLP-модулями підвищує «людяність» бот-інтерфейсу: замість технічних повідомлень бот формує природні відповіді — пояснює причини зміни показників, аналізує дані та дає рекомендації в розгорнутому вигляді. За допомогою ChatGPT-адаптера можна реалізувати діалоги у вільній формі: користувач запитує «Чому вчора було холодніше?» — і система на основі логів відповідає «Температура знизилася вночі через автоматичний зниження опалення в режимі енергозбереження; ви встановили поріг на 18 °C».

Нарешті, важливо зазначити, що інтеграція чат-ботів у ІОТ-системи потребує тестування на стійкість: необхідно моделювати одночасні підключення кількох користувачів, затримки мережі та роботу в умовах відсутності інтернету. Таким чином, чат-бот стає надійним і зрозумілим інтерфейсом для моніторингу та управління домашньою автоматизацією, поєднуючи простоту використання з гнучкістю й потужністю серверної логіки.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ АДАПТИВНОЇ СИСТЕМИ

У сучасних інформаційних та технологічних системах, особливо в межах концепції Інтернету речей (ІОТ) та інтелектуальних екосистем «розумного дому», критично важливим є ретельне та всебічне проєктування архітектури [16]. Саме від продуманої архітектури залежить, наскільки ефективно система збиратиме дані з різнорівневих сенсорів, оброблятиме їх, зберігатиме в історичних архівах та на основі алгоритмів штучного інтелекту ухвалюватиме рішення щодо коригування налаштувань виконавчих механізмів. Основна мета цього розділу — сформулювати сукупність вимог та принципів побудови та інтеграції апаратних і програмних компонентів, забезпечити опрацювання інформаційних потоків у режимі реального часу, а також окреслити модульність і гнучкість системи для подальшого розширення та масштабування.

У ході розробки архітектури поставлено такі ключові завдання:

- визначення апаратної платформи та побудова сенсорного вузла — обґрунтування вибору мікроконтролера, датчиків екологічних параметрів і схеми їх фізичного підключення.
- проєктування комунікаційного шару та аналіз мережевих протоколів — дослідження інтерфейсів на рівні обладнання та протоколів MQTT, HTTP(S), WebSocket з метою оптимального балансу між швидкістю, надійністю та енергоефективністю.
- уніфікація формату повідомлень і побудова системи зберігання даних — розробка єдиного формату серіалізації (JSON), визначення структури локального журналу у контролері та архітектури хмарного сховища для довготривалого аналізу.
- складання структури програмних модулів на рівні збору і попередньої обробки даних — деталізація компонентів, які відповідають за опитування сенсорів, фільтрацію сирих показників і тимчасове кешування.
- проєктування взаємодії компонентів та інформаційних потоків — опис механізмів маршрутизації повідомлень між модулями, координації даних у реальному часі та контролю стану системи.

Для забезпечення високого рівня надійності та відмовостійкості архітектура базується на таких принципах:

- **масштабованість:** можливість легкого додавання нових сенсорів, розгортання додаткових обчислювальних вузлів та збільшення ємності сховища даних без втрати продуктивності існуючих модулів.
- **гнучкість:** компоненти ізольовані одна від одної через чітко визначені інтерфейси (API), що дає змогу модернізувати чи замінювати окремі блоки без повної реконструкції системи.
- **енергоефективність:** реалізація алгоритмів роботи сенсорів і мережевих протоколів з урахуванням обмеженого живлення, перехід у «сплячий» режим між циклами опитування [17].

Наступні підрозділи розділу 2 детально описують кожен із перелічених аспектів.

2.1 Апаратна платформа та сенсорний вузол

Розробка адаптивної системи моніторингу довкілля починається з ретельного вибору апаратної платформи та конфігурації сенсорного вузла. Саме від правильно спроектованої апаратної основи залежить точність вимірювань, надійність збору даних і здатність системи працювати в різних умовах — від житлових приміщень із невеликим споживанням енергії до комерційних чи промислових об'єктів з високими навантаженнями. У даному підрозділі детально розглянуто підхід до формування сенсорного вузла на базі віртуального середовища емуляції Wokwi з використанням плат Arduino, описано ключові датчики, їхні технічні характеристики, принципи підключення й живлення, а також процедури калібрування й тестування, які забезпечують надійність і відтворюваність результатів.

По-перше, у якості апаратної платформи для прототипування обрано емулятор Wokwi API Platform IDE. Цей інструмент дозволяє створювати віртуальні макети на базі плат Arduino Uno або Nano 33 IOT без необхідності фізичного

монтажу компонентів [18]. Візуальний редактор середовища дає змогу «перетягнути» датчики на макетну плату, встановити їхні параметри та одразу запустити скетч у браузері. Такий підхід значно пришвидшує ітерації розробки апаратної складової: не потрібно чекати на доставку деталей, паяти й налагоджувати реальне обладнання — усі перевірки виконуються віртуально, а результати вимірювань логуються в реальному часі.

Основу сенсорного вузла складають три типи датчиків, які разом формують екологічний профіль приміщення:

1. DHT22 (температура і вологість). DHT22 — це популярний цифровий датчик, призначений для вимірювання температури та відносної вологості повітря. Його точність становить приблизно $\pm 0,5$ °C для температури й ± 2 % RH для вологості, що робить його достатньо надійним для загальних завдань моніторингу мікроклімату. Частота оновлення показань обмежена одним зчитуванням раз на дві секунди, але цього вистачає для переважної більшості сценаріїв, де зміна середовища відбувається поступово. Однопровідний інтерфейс з резистором підтягування забезпечує мінімальну кількість з'єднань із платою Arduino та простоту монтажу;

2. MQ-2 (газовий сенсор). MQ-2 — універсальний газовий сенсор, чутливий до чадного газу (CO), летких органічних сполук (VOC), метану, пропану, бутану та диму. Вихідний аналоговий сигнал змінюється в залежності від концентрації газу в повітрі. Оскільки стандартні аналого-цифрові перетворювачі Arduino мають 10-бітну роздільну здатність, рекомендується використовувати зовнішній 12- або 16-бітний АЦП для підвищення точності. Перед початком вимірювань датчик потребує прогріву протягом 20–60 секунд для стабілізації значень. Споживання енергії в робочому режимі може досягати 800 мВт, тому в умовах автономного живлення необхідно передбачити механізм переходу в енергоощадний режим або вимикання живлення, якщо довгий час не відбувається активне опитування.

3. PMS5003 (пиломір). Оптичний сенсор PMS5003 від компанії Plantower використовується для вимірювання концентрацій дрібнодисперсних часток PM1.0, PM2.5 і PM10. Підключається він через інтерфейс UART із бітрейтом 9600 bps і

видає дані в серійному пакеті з вбудованим CRC-контролем, що дозволяє автоматично відкидати пошкоджені кадри. Точність вимірів на рівні $\pm 10\%$ достатня для виявлення загрозливих рівнів забруднення повітря. Щоб уникнути засмічення оптики та потрапляння конденсату всередину сенсора, до корпусу датчика рекомендується додати поліпропіленовий фільтр тонкої очистки [18].

Для коректної роботи сенсорного вузла слід дотримуватися низки рекомендацій щодо підключення:

- DHT22 підключається до цифрового входу плати через резистор підтягування на 4,7 к Ω , що забезпечує стабільний логічний рівень за відсутності комунікації;
- MQ-2 встановлюється на аналоговий вхід, до якого в серійному випадку підключено зовнішній АЦП із буфером на операційного підсилювача, щоб уникнути навантаження на вихід датчика;
- PMS5003 комунікує з платою через USB-UART-конвертер або безпосередньо через відповідні піни TX/RX, при цьому між рівнями 5 V і 3,3 V за потреби ставиться рівневий перетворювач.

Також важливо передбачити захисні елементи:

- розділові резистори та TVS-діоди на входах сигналів запобігають пошкодженню у разі різких перенапруг;
- конденсатори (0,1 μ F і 10 μ F) на лініях живлення згладжують імпульси та зменшують електромагнітні перешкоди;
- при автономному живленні з батареї або сонячної панелі слід реалізувати пріоритетний перехід між джерелами живлення та увімкненням резервного акумулятора за допомогою спеціалізованого контролера заряду (наприклад, TP4056 для літій-іонних акумуляторів) [20].

Щоб забезпечити достовірність показань, для кожного типу датчика реалізуються процедури калібрування:

- DHT22 калібрується на еталонній камері із контрольованою температурою та вологістю. Після серії вимірів у різних точках температурно-вологосної діаграми визначаються коефіцієнти корекції, які зберігаються в пам'яті

контролера. Паралельно застосовується алгоритм ковзного середнього з вікном із 5–10 останніх зчитувань для згладжування випадкових коливань;

- MQ-2 квантується шляхом утворення табличної залежності між відомими концентраціями газу та аналоговими значеннями напруги на виході. Відповідно до цих даних формується перетворювальна функція, яка в реальному часі перераховує сирі значення у ppm (parts per million). Додатково впроваджується фільтр Кальмана або простіше ковзне середнє, щоб зменшити рівень шуму та стрибків;

- PMS5003 не потребує складного калібрування, оскільки поставляється з вбудованим еталоном. Проте рекомендується виконати перевірку точності на побутовому піломірі чи лабораторному обладнанні та в разі потреби коригувати результати програмно [17].

На рівні прошивки перед передачею даних використовується модуль попередньої обробки, що реалізує:

1. визначення та відкидання некоректних пакунків (наприклад, за CRC або вихідних даних у нереальному діапазоні);
2. застосування ковзного середнього для кожного параметра з обраним розміром вікна;
3. формування композитних показників (наприклад, індексу якості повітря AQI на основі комбінації PM2.5 та PM10).

Для тестування апаратної логіки віртуальним середовищем передбачено дві моделі передачі даних:

- серійний інтерфейс (Serial). Передача пакетів у визначеному текстовому форматі зі старт-та стоп-маркерами та контрольними сумами. Такий підхід дозволяє швидко інтегрувати сторонні серійні парсери та перевіряти коректність формування кадрів;

- HTTP REST API. Локальний HTTP-сервер емулятора приймає GET- та POST-запити з даними сенсорів у форматі JSON. Це відокремлює шар збору даних від мережевої логіки та полегшує інтеграцію з автоматизованими тестами на базі скриптів (наприклад, із використанням мов програмування високого рівня чи інструментів на кшталт Postman) [13].

У середовищі Wokwi реалізовано механізми live-логування, що відображають у реальному часі час зчитування, значення кожного датчика та стан вузла. Це дає змогу оперативно виявляти аномалії — наприклад, «зависання» датчика MQ-2 на незмінному рівні чи підвищену вологість без фізичних підстав.

Через зручний GUI можна вручну змінювати параметри датчиків (імітувати спалах чадного газу чи зростання пилу) та спостерігати, як реагує алгоритм калібрування і попередньої обробки. Для перевірки стійкості системи виконуються стрес-тести з різними частотами опитування (від 0,1 Гц до 10 Гц), штучним введенням затримок у передачі (до 2 с) та випадковими втратами пакетів.

Переваги та обмеження підходу з DHT22 замість BME280:

- зниження вартості. DHT22 коштує значно менше, ніж більш функціональний BME280, тому початкові витрати на будівництво кількох вузлів мінімальні;
- простота інтеграції. Однопровідний інтерфейс спрощує монтаж та налаштування прошивки;
- обмежена точність і частота. DHT22 менш стійкий до різких перепадів вологості й температури, а також повільніше оновлює показники у порівнянні з BME280. Щоб компенсувати це, акцент зроблено на якісному калібруванні та алгоритмах обробки даних;
- мінімальні вимоги до пам'яті і ресурсу процесора. Легкість роботи з DHT22 знижує навантаження на мікроконтролер і полегшує розгортання на платах із обмеженими ресурсами.

Таким чином, сенсорний вузол із конфігурацією DHT22, MQ-2 та PMS5003 у поєднанні з віртуальною емуляцією Wokwi створює надійну, гнучку та економічну апаратну платформу. Вона дозволяє швидко прототипувати алгоритми збору й первинної обробки даних, тестувати різні сценарії навантаження та невідкладно виявляти помилки апаратного рівня без додаткових витрат часу й коштів на фізичні компоненти.

У проєкті адаптивної системи моніторингу довкілля реальна фізична збірка апаратури наразі замінена повністю віртуальною емуляцією. Використання Wokwi API Platform IDE дає змогу моделювати поведінку і взаємодію сенсорного вузла без

необхідності фізичного підключення компонентів. Це значно прискорює розробку, дозволяє проводити численні ітерації тестування та одночасно масштабувати кількість віртуальних вузлів без додаткових витрат на “залізо” [12].

Нижче детально описано, як побудований сенсорний вузол у емуляторі, які можливості надає середовище Wokwi, як відбувається конфігурація віртуальних датчиків, їх калібрування, генерування тестових сценаріїв та інтеграція з вищими шарами програмної архітектури.

2.1.1 Переваги емуляторної платформи

У підпункті 2.1.1 «Переваги емуляторної платформи» детально розглядаються ключові аспекти, які роблять використання Wokwi API Platform IDE унікально ефективним інструментом для розробки адаптивних IoT-систем. Акцент зроблено на тому, як віртуальна емуляція прискорює цикл прототипування, спрощує тестування сценаріїв роботи сенсорних вузлів, економить ресурси проєкту і сприяє безперешкодній колаборації в команді [18]. Нижче послідовно розкрито чотири основні напрямки переваг: швидкість прототипування, гнучкість у тестуванні, економія апаратних ресурсів та відтворюваність середовища розробки.

Основні переваги:

1. швидкість прототипування. Запуск нового віртуального вузла вимагає всього двох кліків у візуальному редакторі Wokwi. Конфігурація плати (Arduino Uno чи Nano 33 IoT) та вибір версії ядра Arduino відбуваються у браузері без встановлення додаткових середовищ розробки. Кожна зміна в скетчі компілюється й запускається в лічені секунди, що дозволяє розробникам миттєво перевіряти ідеї й корегувати алгоритми без затримок;

2. гнучкість у тестуванні. Емулятор підтримує одночасну роботу десятків вузлів із різними профілями навколишнього середовища: температура, вологість, наявність газів і концентрація пилу можуть змінюватися у реальному часі. Завдяки панелі налаштувань легко задавати як статичні значення, так і складні анімовані

сценарії коливань. Live-логування та вбудовані графіки забезпечують прозорість даних без необхідності писати окремі скрипти для візуалізації;

3. економія ресурсів. Використання емулятора усуває потребу закуповувати сенсори, плати та інші апаратні компоненти до етапу фінального тестування. Це знижує бюджет проєкту на початкових стадіях і дає змогу сконцентруватися на оптимізації програмної логіки й мережевих протоколів, не витрачаючи кошти на фізичні макети;

4. відтворюваність і колаборація. Конфігурації віртуальних вузлів та сценарії тестів зберігаються у вигляді шаблонів, які легко версіювати в репозиторії. Кожен член команди може імпортувати ці налаштування й однаково точно відтворити середовище колеги, що значно полегшує спільну розробку та пришвидшує виявлення і виправлення помилок [19].

Таким чином, емуляторна платформа Wokwi API Platform IDE надає комплекс переваг, критичних для швидкого та ефективного створення прототипів адаптивних IoT-рішень. Вона зменшує час і витрати на апаратну підготовку, забезпечує гнучкість тестування різноманітних сценаріїв у реальному часі, спрощує процеси налагодження та дозволяє команді легко обмінюватися готовими конфігураціями. Завдяки цим можливостям розробники можуть сконцентруватися на вдосконаленні алгоритмів збору, обробки та аналізу даних, гарантує при цьому високу якість і надійність майбутньої реалізації на реальному обладнанні [16].

2.1.2 Віртуальні датчики в сенсорному вузлі

У даному прототипі сенсорного вузла використовуються три основні віртуальні датчики, які імітують збір екологічної інформації:

1. DHT22 (температура і вологість):

- у вікні налаштувань емулятора можна обрати DHT22 та задати значення початкових умов (наприклад, 22 °C та 45 % RH);
- діапазон змін — від -10 °C до +50 °C та від 0 до 100 % RH, з кроком оновлення до одного зчитування за 2 с;

- для відтворення реалістичних коливань можна налаштувати анімацію: наприклад, плавне зниження температури на 0,5 °C що хвилину або циклічне збільшення вологості.

2. MQ-2 (газовий сенсор):

- емулятор дозволяє задати початковий рівень “газу” в уявних одиницях (наприклад, 50 ppm) та задати сценарій його наростання або різкого спалаху;
- зміна значень може відбуватися лінійно або за наперед заданою траєкторією, що дає змогу тестувати алгоритми виявлення аварійних ситуацій (різке зростання шкідливих випарів).

3. PMS5003 (концентрація пилу):

- налаштування концентрацій PM1.0, PM2.5 та PM10 виконується через спеціальне вікно емулятора, де вказуються початкові значення і варіанти їх зміни;
- емулятор реалізує пакети даних із CRC-контролем, що дозволяє відтворити поведінку справжнього датчика та перевірити коректність обробки неповних чи пошкоджених кадрів.

Завдяки таким можливостям Wokwi API Platform IDE забезпечує реалістичну емуляцію сенсорного вузла й швидку перевірку алгоритмів збору та обробки даних.

2.1.3 Конфігурація емулятора та віртуальних зв'язків

Хоча сенсорні датчики в Wokwi фізично “не підключаються” до Arduino, середовище імітує логіку зв'язку через віртуальні піни:

- у графічному інтерфейсі потрібно просто запрограмувати значок датчика до віртуальної плати, призначити йому номер цифрового або аналогового входу;
- емуляція такої прив'язки гарантує, що скетч «бачить» датчик так, ніби той реально підключений до обраного піна;
- після запуску Arduino IDE збірка підключень та налаштувань зберігається у файлі конфігурації, що передається разом із кодом для наступних запусків.

Цей віртуальний інструментарій настільки достовірно моделює апаратні взаємодії, що жодних додаткових відомостей про фізичні резистори, розділювачі чи діоди вводити не потрібно. Інформація про електричні захисні заходи в цьому режимі застаріла — всі такі аспекти вже враховані всередині емулятора.

2.1.4 Калібрування та точність вимірювань у віртуальному середовищі

Незважаючи на відсутність справжніх сенсорів, емулятор надає можливість відтворити процес калібрування:

- налаштування точності. Для кожного датчика в панелі властивостей можна вказати похибку (наприклад, $\pm 0,5$ °C для DHT22 або ± 10 % для PMS5003). Емулятор додаватиме випадковий шум згідно з цими параметрами, щоб моделювати реальні умови роботи;
- автоматичне виведення статистики. Після запуску тесту вбудовані графіки показують середнє значення та розсіювання вимірювань. Це допомагає налаштувати алгоритм згладжування або фільтрації, не виходячи із браузера;
- сценарії калібрування. Можна створити кілька етапів калібрування з різними вхідними умовами (наприклад, низька температура та висока вологість, а потім навпаки) і перевірити, як обрана методика корекції (ковзне середнє, фільтр Кальмана) реагує на зміну параметрів.

Отже, навіть у повністю віртуальному середовищі Wokwi забезпечує достатній рівень реалістичності для побудови й перевірки методик калібрування сенсорів. Налаштування вбудованої похибки і генерація випадкового шуму дають змогу відтворити ключові характеристики реальних датчиків, а автоматичне виведення статистики в реальному часі спрощує вибір і тонке налаштування алгоритмів згладжування чи застосування фільтрів.

Реалізація сценаріїв калібрування з різними граничними умовами дозволяє переконатися в коректності та стійкості обраних методик корекції показань за широкого спектру змін довкілля. У результаті, розробники отримують гнучкий інструмент для оцінки точності вимірювань і відпрацювання алгоритмів обробки

даних без потреби у фізичному стенді, що значно пришвидшує цикл розробки та підвищує надійність майбутніх експериментів із реальними сенсорними вузлами.

2.1.5 Інтеграція з програмною архітектурою

Після того, як дані з віртуального вузла сформовані й протестовані, наступний крок — передача їх до сервісів мережевого шару:

- серійний інтерфейс в скетчі Arduino залишається активним, і дані виводяться у форматі, який обробляє локальний парсер передавання;
- REST-запити до локального емулятора дозволяють тренувати серверну частину: запити на отримання останнього зчитування та надсилення масивів даних;
- відпрацьовані в емуляторі шаблони пакетів і форматів повідомлень можна без змін перенести на реальну апаратну платформу, адже їхня специфікація залишається ідентичною.

Саме завдяки емулятору розробники отримують єдиний набір віртуальних інтерфейсів, який у подальшому легко масштабувати й переносити на реальний “залізний” вузол без додаткових змін у коді чи структурі повідомлень.

2.1.6 Підсумки емуляції сенсорного вузла

У підрозділі 2.1 було показано, що віртуальна емуляція сенсорного вузла на базі Wokwi API Platform IDE — це не просто тимчасове рішення для проектування, а потужний інструмент, який дозволяє:

- швидко прототипувати апаратну частину без фізичного монтажу;
- гнучко налаштовувати та калібрувати віртуальні датчики;
- створювати реалістичні тестові сценарії аварій та навантажень;
- відтворювати середовище кожного вузла в команді розробників;
- інтегруватися з мережею та вищими модулями без додаткової адаптації;

- отримувати деталізовані логи та графіки прямо в браузері.

Цей підхід створює надійну основу для подальшої розробки мережевого шару, форматів обміну даними та аналітичних сервісів.

2.2 Комунікаційний шар і мережеві протоколи

Комунікаційний шар відповідає за передачу даних між сенсорним вузлом на базі Arduino (або його емуляцією в Wokwi) та серверною частиною, де здійснюється зберігання, аналіз і взаємодія з користувачем. У нашій системі застосовано три взаємодоповнюючі протоколи — MQTT, HTTP та WebSocket — кожен із яких оптимізований під певні типи повідомлень і сценарії роботи [5].

MQTT (Message Queuing Telemetry Transport) використовується для регулярної публікації коротких повідомлень із показниками датчиків. Завдяки моделі «publish/subscribe» та низькій «вазі» заголовків MQTT забезпечує мінімальне мережеве навантаження, а механізми QoS дозволяють гарантувати доставку критично важливих даних навіть у разі тимчасових збоїв з'єднання. Топіки організовано за ієрархією (наприклад, home/livingroom/temperature), що спрощує масштабування — нові вузли одразу публікують у свої підтематики [5].

HTTP/REST застосовується для передавання об'ємних пакетів даних, зокрема CSV-файлів із накопиченими логами. Одноразовий POST-запит дозволяє надіслати архів за певний інтервал або за командою користувача. Це зручно для інтеграції з веб-хуками (webhooks) та сторонніми сервісами, які не підтримують MQTT [5].

WebSocket створює двосторонній канал «у режимі реального часу» між сервером і чат-ботом. За допомогою WebSocket надсилаються негайні аварійні сповіщення та оновлення інтерфейсу — наприклад, коли рівень диму перевищує поріг, бот миттєво отримує подію й оповіщає користувача.

Поєднання цих протоколів забезпечує гнучкість: MQTT відповідає за постійну телеметрію, HTTP — за пакетну передачу великих фрагментів даних, а WebSocket — за оперативні двосторонні повідомлення. Такий підхід гарантує

надійність, масштабованість і своєчасність доставки інформації в різних робочих умовах [5].

2.3 Формат і сховище даних

Для первинного зберігання телеметрії від сенсорного вузла обрано простий та універсальний формат CSV (Comma-Separated Values). Кожен рядок CSV-файлу відповідає одному пакету вимірів і складається з полів:

- `timestamp` — часовий штамп у форматі ISO 8601 (YYYY-MM-DD HH:MM:SS), який дозволяє коректно впорядковувати записи та виконувати часові запити;
- `sensor ID` — унікальний ідентифікатор датчика або групи датчиків (наприклад, DHT22_1, MQ2_3), що спрощує фільтрацію й аналіз окремих потоків даних [6];
- `raw value(s)` — сирі показники, зняті без попередньої обробки (значення АЦП або цифровий код), необхідні для повторного калібрування або діагностики похибок;
- `calibrated value(s)` — обчислені за внутрішніми алгоритмами температури, вологості, концентрації пилу або диму, готові до безпосереднього аналізу;
- `status Code` — службове поле для повідомлення про помилки чи стан вузла (наприклад, OK, SENSOR_ERROR, LOW_BATTERY).

Завдяки CSV-файлам ми отримуємо кілька важливих переваг:

- швидке відлагодження — будь-який текстовий редактор або табличний процесор (Excel, LibreOffice Calc) дозволяє відкрити файл і миттєво побачити структуру даних;
- універсальна сумісність — CSV підтримують практично всі мови та бібліотеки для роботи з даними (Python's pandas, R, MATLAB), що полегшує розробку скриптів аналітики;

- мінімальні вимоги до інфраструктури — відсутність необхідності налаштовувати СУБД або окремий сервер зберігання робить систему простою у розгортанні та обслуговуванні.

Файли зберігаються на диску сервера в директорії logs із іменами, що відповідають діапазонам дат, наприклад data_2025-05-01_2025-05-07.csv. Кожного дня автоматичний скрипт перевірки створює новий файл із відповідним часовим проміжком, а старі архівуються у формат ZIP для зменшення обсягу. За потреби передбачено простий механізм ротації: якщо загальний розмір каталогу logs/ перевищує заданий ліміт (наприклад, 1 ГБ), найстаріші ZIP-архіви видаляються, зберігаючи лише останні 30 днів даних.

Такий підхід до формату і сховища даних забезпечує баланс між простотою реалізації та гнучкістю подальшої обробки: початковий CSV використовується для швидкого тестування й візуального аналізу, а за необхідності його можна легко імпортувати в більш складні сховища чи аналітичні конвеєри без втрати інформації.

2.4 Архітектура програмних модулів

Програмна частина системи розділена на п'ять взаємопов'язаних модулів, кожен із яких відповідає за окремий шар функціональності. Така структура дозволяє працювати над кожним елементом незалежно, легко підключати тести та за необхідності замінювати реалізації.

1. sensor_reader:

- написаний на Python із використанням бібліотеки `pyserial` (для реального Arduino) або HTTP-клієнта (для Wokwi API);
- періодично опитує інтерфейси I²C/аналогові входи, перетворює сирі дані в числові значення, проводить просту перевірку на коректний діапазон і формує JSON-повідомлення з полями `sensor\_id`, `timestamp`, `raw\_value`.

2. comms_adapter:

- спроектований за патерном «стратегія»: підтримує три підкласи – MQTT, HTTP і WebSocket – які реалізують єдиний інтерфейс `send(data)` та подієвий метод `on_receive()` [5];

- при ініціалізації вибирає потрібний конектор на основі конфігурації, автоматично підключається до брокера або відкриває сокет, відновлює з'єднання в разі таймауту.

3. storage_manager:

- використовує асинхронні черги (`asyncio.Queue`) для прийому повідомлень від `comms_adapter` [9];

- записує дані в CSV у фоновому режимі, слідкує за максимальною довжиною файлу та розбиває його на щоденні частини. Реалізовано обробку помилок диску (I/O exceptions) і створення резервної копії на випадок некоректного завершення запису.

4. ai_adapter:

- формує людськомовні запити (prompts) на основі шаблонів і збережених показників;

- надсилає їх до OpenAI API через офіційну бібліотеку `openai`, чекає відповіді, аналізує JSON-відповідь і перетворює її на об'єкт із полями `recommendation_text`, `confidence_score` [10].

5. bot_interface:

- побудований із використанням бібліотеки `python-telegram-bot` [11];
- реєструє обробники команд (`Показати дані`, `Показати графіки`, `recommend`), отримує дані від інших модулів через спільну подієву шину, надсилає фінальне повідомлення користувачу з Markdown-розміткою та прикріпленням графіком.

На рисунку 2.1 зображено загальну архітектуру рішень.



Рисунок 2.1 – Діаграма архітектури рішень проекту

1. Ардуіно-/Wokwi (sensor_reader) – фізичний сенсорний вузол або його емуляція, що генерує «сирі» дані.
2. Сервіс прийому даних (storage_manager) – приймає JSON-повідомлення від sensor_reader через comms_adapter та зберігає їх у CSV-файли.
3. Модуль обробки та візуалізації (data_processor) – зчитує CSV, обчислює агреговані метрики, створює графіки та формує звіти у форматі CSV.
4. Модуль інтелектуального аналізу (ai_adapter) – надсилає «людськомовні» запити до ChatGPT через OpenAI API, отримує рекомендації і зберігає їх у окремий звітний CSV.
5. Звіти та інформаційні панелі – вихідні файли CSV і PNG-графіки, які можна переглядати поза системою або передавати далі (наприклад, у BI-інструменти).
6. Сервіс сповіщень (bot_interface) – Telegram-бот, що приймає команди від користувача та надсилає готові дані, графіки й рекомендації.

Завдяки такій модульній архітектурі:

- кожний компонент можна протестувати та розвивати окремо;

- за потреби можна змінити `comms_adapter` (наприклад, перейти з MQTT на повністю HTTP-підхід) без значних переробок у інших модулях;
- легко підключити новий модуль обробки (наприклад, веб-інтерфейс замість Telegram) — достатньо реалізувати аналогічний інтерфейс `send(data)` чи обробник команд.

Міжмодульна взаємодія відбувається через асинхронні події та черги, що гарантує неминучість доставки повідомлень навіть під час пікового навантаження. Логування ведеться централізовано із рівнями `DEBUG`, `INFO`, `ERROR`, що спрощує відстеження помилок і продуктивності кожного модуля. Такий підхід робить архітектуру прозорою, гнучкою для розширення та простою для підтримки.

2.5 Взаємодія компонентів та інформаційні потоки

У запропонованій архітектурі всі компоненти працюють в єдиному асинхронному середовищі, обмінюючись даними через подієві черги та легковагові повідомлення. Загальна послідовність інформаційного потоку виглядає так:

1. зняття даних;
2. ``sensor_reader`` опитує емуляцію Arduino у Wokwi або реальний пристрій, отримує сирі значення з датчиків і формує внутрішні повідомлення з полями ``sensor_id``, ``timestamp``, ``raw_value``;
3. передача повідомлення. Замість прямого виклику між компонентами, ``sensor_reader`` публікує повідомлення у внутрішню чергу або надсилає через ``comms_adapter`` на MQTT-брокер (топик ``sensors_data``), HTTP-ендпоінт чи WebSocket-канал [5];
4. розгалуження потоків. `Storage_manager` підписується на той самий топик і асинхронно записує отримані пакети в CSV-файл, забезпечуючи безперервне логування. `Ai_adapter` також отримує копію повідомлення (з тією ж затримкою QoS) і накопичує дані для формування промптів до OpenAI API [10];

5. генерація рекомендацій. Після надходження достатнього обсягу записів ``ai_adapter`` формує запит до ChatGPT, отримує текстову відповідь і публікує її у внутрішній топик ``ai_recommendations``;

6. взаємодія з користувачем. ``bot_interface`` стежить за двома джерелами подій: прямими командами (``Показати дані``, ``Показати графіки``, ``recommend``) від користувача та подіями з топиків (``sensors/data`` для аварій, ``ai_recommendations`` для звітів). Відповідно до типу події він формує повідомлення Telegram із графіками або текстовими висновками;

7. обробка помилок і логування. Всі модулі пишуть логи з рівнями INFO/ERROR у загальний файл, а в разі відмови з'єднання ``comms_adapter`` автоматично перепідключається.

Такий поділ ролей і чітка маршрутизація даних через топіки гарантують, що жоден етап обробки не блокуватиме інший, а система залишиться стійкою до збоїв і легкою для масштабування.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі детально розглядається практична реалізація програмної частини адаптивної системи домашньої автоматизації, яка поєднує збір даних із сенсорного вузла, їхнє надійне збереження, інтелектуальну аналітику на базі ChatGPT і зручний інтерфейс взаємодії через Telegram-бот. Основна ідея полягає в створенні наскрізного інформаційного потоку — від фізичного виміру температури, вологості та якості повітря до напрацьованих рекомендацій і керуючих команд. Для цього використано модульний підхід, де кожна підсистема відповідає за чітко визначений етап: зчитування та обробка «сирих» даних, їхнє логування й публікація, генерація адаптивних висновків і, нарешті, комунікація з користувачем.

У якості основної мови програмування обрано Python. Він надає потужний набір бібліотек для роботи з асинхронним I/O, мережею та обробкою даних, що особливо важливо для ІОТ-систем із великою кількістю одночасних запитів. Асинхронний фреймворк ``asyncio`` дозволяє розв'язувати проблему блокувань у процесах зчитування сенсорів, запису на диск та комунікації з віддаленими сервісами. Кожен із ключових компонентів працює як окремий асинхронний сервіс — так званий «мікросервіс в одному процесі», що забезпечує незалежність розробки та тестування. Такий підхід дає можливість у майбутньому легко масштабувати систему, додаючи нові вузли без суттєвих змін у решті коду [9].

Процес розробки розпочався зі створення модуля зчитування даних із сенсорів, який покликаний перетворити «сирі» байти або HTTP-запити від емулятора у структуровані дані. Надалі ці дані потрапляють у підсистему зберігання, де реалізовано гнучкий механізм логування в CSV-файли та одночасну публікацію в MQTT-топіки й HTTP-ендпоінти. Ротація файлів, архівація старих даних та контроль обсягу сховища гарантують довготривалу стабільну роботу навіть на обмежених ресурсах. У результаті всі показники доступні для подальшого аналізу й відтворювані для ретроспективного дослідження [5].

Однією з ключових інновацій є інтеграція з OpenAI API. Спеціальний модуль вибирає найбільш релевантний набір даних за заданий інтервал, формує природномовний опис трендів та надсилає запит до GPT-моделі. Отримана у відповідь інформація обробляється й конвертується у чіткі рекомендації із зазначенням пріоритетності дій. Такий підхід дозволяє вийти за межі «сухої» статистики: користувачі отримують зрозумілі пояснення причин змін клімату в приміщенні і поради, як оптимізувати споживання енергії та підвищити безпеку [10].

Останній етап — взаємодія з користувачем — реалізовано через Telegram-бота. Завдання цього компонента — зробити взаємодію максимально інтуїтивною: підписка на аварійні сповіщення, отримання стислих статусних звітів, генерація графіків за запитом і надсилання адаптивних рекомендацій. Команди бота побудовані за принципом «одна команда — одна дія», що полегшує користувачу швидко отримати потрібну інформацію без зайвих зусиль. Завдяки такому інтерактивному інтерфейсу система стає доступною навіть тим, хто не має технічної підготовки, і дозволяє контролювати домашню автоматизацію з будь-якої точки світу.

Насамкінець, реалізація програмного забезпечення проведена з акцентом на модульність, стійкість до збоїв та гнучкість архітектури. Кожен сервіс логічно відокремлений, має чіткий API і власну систему логування. Це спрощує налагодження, тестування та впровадження нових можливостей. Завдяки використанню відкритих стандартів і загальновживаних форматів даних, проєкт може бути легко адаптований під інші сценарії — від моніторингу промислового обладнання до роботи в смарт-офісах.

3.1 Модуль зчитування і попередньої обробки даних із датчиків

Модуль зчитування виступає «точкою входу» для всіх даних із сенсорів. У першу чергу він забезпечує надійну та регулярну взаємодію з вузлом Arduino (або його емуляцією у Wokwi). В інтервалах, заданих у конфігураційному файлі, сервіс

ініціює запити до послідовного порту або HTTP-ендпоінта емулятора, отримує сирі показники і переводить їх у фізичні одиниці.

Низькорівнева обробка складається з кількох етапів.

Калібрування: кожен тип сенсора має власні конверсійні формули й коефіцієнти, що зберігаються у налаштуваннях. Наприклад, для датчика DHT22 температура обчислюється за формулою виробника з урахуванням множинних поправок, а для оптичного сенсора пилу застосовується корекція за температурою й вологістю [6].

Валідація: отримані значення перевіряються на відповідність очікуваним діапазнам. У разі виходу за межі модуль фіксує статус помилки та викликає механізм аварійного оповіщення.

Фільтрація: для боротьби зі стрибкоподібними значеннями використовується алгоритм ковзного середнього або експоненційного згладжування. Параметри фільтра задаються окремо для кожного типу сенсора, що гарантує збереження динаміки тренду при видаленні випадкових сплесків [20].

Формування уніфікованого повідомлення: після перетворень модуль збирає результат у внутрішній формат даних, що включає часову мітку, ідентифікатор вузла, сирі та фільтровані значення і статус. Така структура полегшує подальшу обробку в наступних шарах архітектури.

Усі операції розгорнуті в асинхронному циклі, що дозволяє модулю коректно працювати навіть при тимчасовій недоступності вузла чи затримках мережі. Логування кожного етапу (успішні зчитування, помилки валідації, обробка винятків) забезпечує швидку діагностику й пришвидшує пошук несправностей.

3.1.1 Архітектура та налаштування модуля

Модуль побудовано за принципом конфігурованих підключень, що описуються зовнішнім файлом конфігурації. У ньому визначено часові інтервали опитування, метод взаємодії з вузлом (через серійний інтерфейс або HTTP-запити до емульованого пристрою), параметри таймаутів і політики повторення у разі

помилки. Окремі налаштування стосуються кожного сенсора: ідентифікатор піна, мінімальні та максимальні допустимі значення, а також параметри фільтрації та обробки шуму. Такий підхід дає змогу гнучко адаптувати модуль до різних апаратних конфігурацій або переходити від емуляції віртуальних датчиків до реального обладнання без змін у коді [7].

Загальна структура модуля передбачає створення основного циклу, який у встановлені інтервали ініціює процедуру зчитування. Цей цикл є асинхронним, що дозволяє паралельно обслуговувати декілька сенсорних вузлів та інших внутрішніх процесів (наприклад, відправлення даних, обробка аварійних станів). Відмовостійкість забезпечується політиками повторної спроби підключення, адаптивним збільшенням затримок при збої та можливістю автоматичного переходу вузла у режим технічного обслуговування [9].

3.1.2 Валідація отриманих даних

Після отримання сирих показників модуль проводить їхню валідацію за низкою правил. Перший рівень включає перевірку наявності всіх необхідних полів та коректність типів даних. Далі слідують діапазонні контрольні точки, що визначають допустимі межі для кожного параметра (температура, вологість, рівень диму, пилу). У разі виходу значення за межі модуль фіксує статус помилки та вмикає механізм аварійного оповіщення — користувач отримує повідомлення про некоректність вимірювання, а система намагається повторити зчитування. У разі багатократних невдалих спроб вузол позначається як тимчасово недоступний, і логіка опитування переходить до наступних зареєстрованих вузлів. Такий підхід гарантує захист від артефактів, збоїв живлення чи коротких наводок.

Поряд з базовими перевітками в модулі передбачено динамічну зміну порогів валідації залежно від режиму роботи. Наприклад, при тестуванні у віртуальному середовищі вимоги можуть бути жорсткішими, а в польових умовах — допускається більша похибка. Це забезпечується налаштуванням гнучких профілів валідації у конфігураційному файлі.

3.1.3 Фільтрація і згладжування даних

Сировинні сенсорні сигнали часто містять короточасні стрибки та шуми. Для їхнього усунення застосовуються два основних підходи: ковзне середнє значення та згладжування на основі експоненційної залежності. Параметри фільтрів (довжина вікна обчислення, коефіцієнти згладжування) можна задавати окремо для кожного датчика. Після кожного етапу фільтрації модуль порівнює вихідний та згладжений сигнали за граничною різницею — якщо зміна надто суттєва, це вказує на ймовірний сплеск чи збій, і дані знову верифікуються або повторно фільтруються іншим алгоритмом. Такий багаторівневий підхід дозволяє зберігати репрезентативність трендів у довгостроковій перспективі та одночасно усувати короточасні нетипові коливання [8].

Крім того, модуль підтримує умовний перехід між видами фільтрації в залежності від оцінки поточної якості сигналу. Якщо система виявляє перевищення допустимого рівня шуму у стандартизованих метриках, вона може автоматично перейти на більш агресивний алгоритм згладжування або активувати додаткові корекційні процедури.

3.1.4 Формування єдиного повідомлення для наступних етапів

Після завершення попередньої обробки дані формуються в єдину структуру, яка включає часову мітку з високою роздільною здатністю, ідентифікатор вузла, первинні сирі значення та їхні згладжені аналоги, а також код статусу (успішно опитано, помилка валідації, вузол недоступний). Цей пакет інформації упаковується у легкий текстовий або бінарний формат і передається далі на сервіси збереження та аналітики. Така уніфікована структура спрощує інтеграцію з різними підсистемами: базою даних, брокером повідомлень, аналітичними модулями.

3.1.5 Асинхронна обробка і відновлення після збоїв

Модуль працює в рамках єдиного асинхронного циклу, що дозволяє не блокувати інші процеси при очікуванні відповіді від пристрою. У разі відмови будь-якої частини ланцюга (ні одне опитування не завершилося успішно, відсутні дані довший час) модуль ініціює процедуру відновлення: перезапускає підсистему зв'язку, змінює налаштування інтервалів опитування та за необхідності переводить вузол у стан технічного обслуговування. Відсутність блокувань гарантує, що навіть при проблемах з одним вузлом інші датчики продовжують працювати без затримок.

Вбудовані таймери та механізми відкладеного повтору сприяють адаптивному реагуванню на різні умови: від короткочасних втрат зв'язку до тривалих перебоїв з харчуванням.

3.1.6 Логування модуля та спостереження за здоров'ям

Для швидкої діагностики та моніторингу роботи модуля передбачено розширене логування: усі успішні зчитування та критичні події фіксуються на різних рівнях важливості (інформаційні повідомлення, попередження, помилки). Логи ведуться у структурованому форматі, що спрощує їхній аналіз за допомогою спеціалізованих систем моніторингу та аналізу логів.

Паралельно із цим модуль експортує ключові метрики — загальну кількість опитувань, відсоток невдалих опитувань, середній час обробки одного циклу — у систему спостереження. Це дозволяє у реальному часі відслідковувати продуктивність, виявляти аномалії та налаштовувати оповіщення, якщо показники виходять за нормальні межі[14].

Кожне сповіщення про збої або відхилення передається на центральний моніторинговий сервер і може ініціювати автоматичну реакцію — наприклад, перезапуск модуля, відправлення SMS чи email адміністратору.

3.2 Зберігання та публікація даних

Після первинної обробки повідомлення потрапляють до підсистеми збереження та публікації. Вона складається з двох взаємодоповнюючих частин: локального запису у CSV та віддаленої публікації через MQTT і HTTP [5].

- Локальне збереження:

дані групуються за годинами або за досягненням ліміту рядків і записуються у файл CSV із добре визначеною структурою колонок: `timestamp`, `sensor_id`, `raw_value`, `processed_value`, `status`. Для уникнення блокування дискового I/O використано буферизацію та фонову чергу повідомлень. Файли розділені за часовими проміжками, що дозволяє швидко завантажувати потрібний інтервал для аналізу. Регулярна архівація (ZIP) та ротація старих файлів забезпечують підтримання обсягу сховища в лімітах.

- Віддалена публікація:

MQTT: кожне повідомлення публікується під темою `home/sensors/telemetry`. Завдяки обраному рівню QoS 1 система забезпечує мінімальний накладний трафік і одночасно гарантує доставку. MQTT-брокер може бути як локальним (Mosquitto), так і хмарним[5].

HTTP: для пакетної передачі повних CSV-файлів або великих об'ємів даних реалізовано REST-інтерфейс. Після годинної архівації файл передається методом POST на ендпоінт `/upload`, де сервер приймає, розпаковує й зберігає архів.

Таке поєднання локального та віддаленого зберігання гарантує надійність: навіть у разі відсутності з'єднання дані не втрачаються, а після відновлення каналів миттєво передаються на сервер.

3.3 Інтеграція з OpenAI API для аналізу трендів і генерації рекомендацій

Інтелектуальний шар адаптивної системи реалізовано за допомогою інтеграції з OpenAI API, що дозволяє автоматизувати перетворення числових даних у зрозумілі для людини пояснення та практичні рекомендації. Цей підпункт описує ключові етапи аналітичного конвеєра — від вибірки даних до доставки готового висновку.

Аналітичний сервіс працює за розкладом або за вимогою користувача. При плановому запуску він звертається до сховища, збирає всі записи з CSV файлу за заданий проміжок (за замовчуванням 24 години). Для мінімізації ресурсного навантаження передбачено попередню фільтрацію: вибираються тільки останні значення кожного півгодинного інтервалу або інтервалу, заданого політикою агрегації. Це скорочує обсяг вхідного масиву даних без втрати суті тренду.

Крім базових показників (температура, вологість, дим, пил), до набору можуть входити статистики як-от:

- Медіана та розмах значень за добу;
- Кількість сплесків (показників за межею стрибкоподібної зміни);
- Середній часовий інтервал між аварійними спалахами;
- Тривалість та частота періодів стабільної роботи.

Ці дані збираються у вигляді ключ-значення, що дозволяє легко формувати запит до моделі без зайвих обчислень на її стороні.

Зібрана статистика автоматично перетворюється в компактний опис трендів. Приклади повідомлень:

"Заклади кімната протягом останніх 6 годин зберігала середню температуру 22,8 °C із піками до 27,5 °C біля опівдня."

"Рівень пилу протягом ночі тримався в межах 0,01–0,03 mg/m³, проте зранку спостерігався короткочасний стрибок до 0,12 mg/m³."

У prompt додаються фрагменти, що позначають критичні ситуації, наприклад:

"За останню добу зафіксовано три випадки підвищення концентрації диму понад 350 ppm на термін понад 5 хвилин."

Цей текст доповнюється метаданими про міський клімат, сезонні особливості або внутрішні події (якщо такі відомі), що підсилює контекст для AI-моделі.

Отриманий prompt передається до GPT через метод API completion або chat completion. Конфігуровані параметри включають:

- Temperature та top_p для контролю креативності відповіді;
- максимальну довжину (token limit) з урахуванням граничних умов;
- System prompt, що задає роль: наприклад, "Ти — розумний асистент з побутової екології, твоє завдання — пояснювати кліматичні зміни й давати практичні поради".

Відповідь AI може містити кілька розділів:

1. опис тренду — резюме змін у цифрах;
2. причини — коротке обґрунтування: «Це може бути пов'язано з інтенсивною роботою опалення»;
3. рекомендації — перелік дій із позначенням пріоритетів;
4. попередження — якщо виявлено ризики для здоров'я [19].

Після отримання текст нормалізується: видаляються зайві вступи, узагальнення структуруються у перелік, категорії маркуються тегами (наприклад, [ENERGY], [HEALTH], [SECURITY]).

Готовий набір порад зберігається в таблиці recommendations з такими полями:

- timestamp — дата та час аналізу;
- prompt_summary — коротка підсумкова фраза запиту;
- full_text — повний текст відповіді AI;
- priority — внутрішня шкала 1–5;
- status — «нове», «відправлене користувачеві», «заплановане».

Користувачі можуть переглядати історію рекомендацій, фільтрувати їх за пріоритетом або темою. Автоматичний механізм оновлення дозволяє відправляти нагадування через заданий інтервал.

З метою захисту ключів та контролю витрат передбачено:

- шифрування секретів у Vault або подібній системі;
- ліміти на кількість токенів на місяць і на запит;
- обробку помилок типу rate limit або network errors з трьома спробами та експоненційним бекофом;
- логування витраченої кількості токенів для кожного аналізу.

3.4 Розробка чат-бота Telegram для відображення графіків і взаємодії з користувачем

Telegram-бот виступає основним інтерфейсом для кінцевого користувача, забезпечуючи зручний та одразу зрозумілий доступ до поточних показників, трендів та рекомендацій. Він реалізований із використанням бібліотеки `python-telegram-bot` і працює за моделлю long polling або webhook, залежно від конфігурації сервера [11].

Основні команди бота:

«Показати дані» — повертає останні показники усіх сенсорів у текстовому вигляді, використовуючи Markdown для форматування та емоджі для індикаторів. Інформація відображає час зчитування, значення температури, вологості, пилу та диму, а також статус системи (OK або ALERT);

«Показати графіки» — генерує графік зміни обраного сенсора за зазначений проміжок часу. Бот викликає `storage_manager` для завантаження даних із CSV, створює графік за допомогою `matplotlib` чи подібного інструменту, зберігає зображення та надсилає його користувачу як фото;

`recommend` — надсилає останній набір рекомендацій, отриманих від ChatGPT через модуль `ai_adapter`. Бот читає повідомлення з топика `home/ai/recommendations` та виводить зведений підсумок, детекцію аномалій і перелік дій.

Крім ініційованих користувачем запитів, бот також отримує сповіщення про аварійні ситуації. Він підписаний на MQTT-топик `home/alerts` і відразу надсилає

“push”-повідомлення з коротким описом проблеми: « Alert: рівень диму перевищено! Рекомендації надсилаються окремо» [5].

Для кращого розуміння роботи чат-боту, було створено діаграму послідовності що зображена на рисунку 3.1.

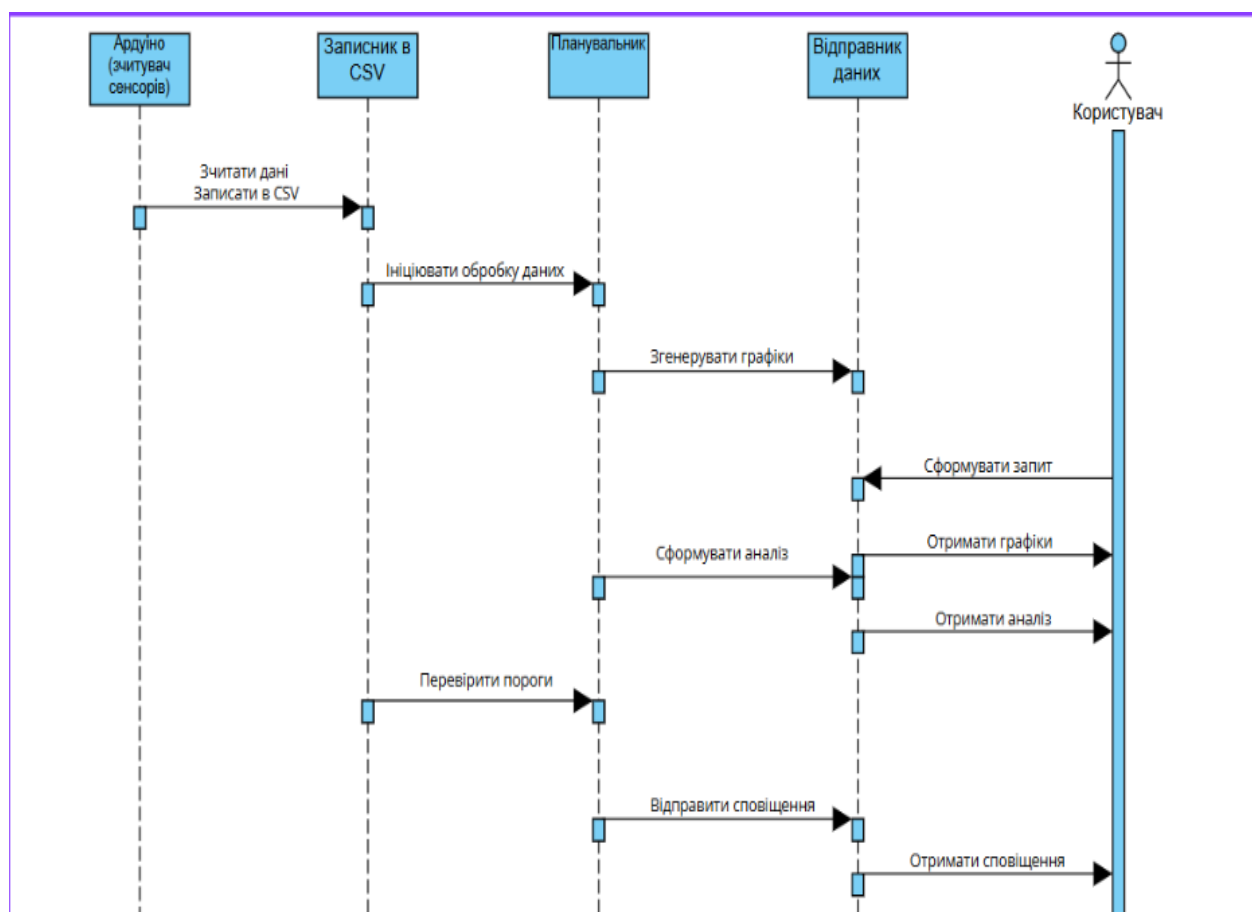


Рисунок 3.1 – Діаграма послідовностей чат боту

1. Ардуіно (зчитувач сенсорів) – записник в CSV:
 - модуль `sensor_reader` (позначений як «Ардуіно (зчитувач сенсорів)») періодично зчитує значення з датчиків (DHT22, MQ-2, PMS5003);
 - після кожного зчитування формує JSON-пакет із полями `sensor_id`, `timestamp` та `raw_value` і передає його до модуля `storage_manager`, який одразу записує дані у CSV-файл.
2. Записник в CSV – планувальник:

- як тільки новий рядок з'являється у CSV, модуль `storage_manager` надсилає внутрішній сигнал (через `asyncio`-чергу) до модуля `data_processor` («Планувальник»), що означає: «Дані оновлено — можна починати обробку».

3. Планувальник – відправник даних (згенерувати графіки):

- модуль `data_processor` читає CSV-файл, використовуючи `pandas`, виконує агрегацію за вказаний інтервал (наприклад, 1–24 години) та будує графіки за допомогою `matplotlib`;

- після створення PNG-файлів із графіками він передає команду «Згенерувати графіки» до модуля `bot_interface` («Відправник даних»), надаючи посилання або самі зображення.

4. Планувальник – Планувальник (сформувати аналіз та перевірити порогови):

- паралельно з побудовою графіків `data_processor` виконує перевірку значень сенсорів на відповідність встановленим порогам (наприклад, рівень диму $> 800 \text{ raw}$ або концентрація пилу $> 0,2 \text{ mg/m}^3$);

- якщо виявлено аномалію, модуль формує внутрішній запис «Аномалія» та передає його собі ж для подальшої обробки («Сформувати аналіз»).

5. Планувальник – відправник даних (відправити сповіщення):

- у разі перевищення порогу `data_processor` надсилає до `bot_interface` команду «Відправити сповіщення» з коротким текстом « Аномалія: рівень диму перевищено» або аналогічним;

- модуль `bot_interface` миттєво розсилає аварійне push-повідомлення до Telegram-клієнта користувача.

6. Користувач - відправник даних (сформувати запит):

- окремим потоком, коли користувач відправляє команду `/status`, `/graph` або `/recommend`, Telegram-бот («Відправник даних») приймає запит;

- бот формує внутрішній запит до `data_processor` або `ai_adapter`:

- 1 для `/status` – запит «Отримати останні значення з CSV»;

- 2 для `/graph` – запит «Отримати графіки за вказаний інтервал»;

- 3 для `/recommend` – запит «Отримати останні рекомендації від ChatGPT».

7. Відправник даних – користувач (отримати графіки / отримати аналіз / отримати сповіщення):

- після того як `data_processor` віддає готові графіки або `ai_adapter` повертає текст рекомендації, `bot_interface` формує відповідь у Markdown-форматуванні (з емоджі та розміткою) і надсилає її користувачеві.

Таким чином, наведена діаграма демонструє ключові кроки, які відбуваються від моменту зчитування даних із датчиків до отримання кінцевим користувачем повного звіту — будь то текстові показники, гнучкі графіки або “тривожне” автоматичне сповіщення про аномалію.

Усі дії бота логуються з рівнями INFO (звичайні запити), WARNING (тривожні події) і ERROR (збої в обробці). Лог-файл бот-сервісу містить часові мітки команд, ідентифікатори користувачів та повідомлення про помилки, що дозволяє адміністраторам оцінити навантаження та швидко реагувати на неполадки.

Для безпеки бот підтримує механізм базової автентифікації: лише попередньо зареєстровані користувачі можуть виконувати команди, а критичні дії (наприклад, зміна налаштувань системи) доступні лише розробнику.

Завдяки такому детальному підходу Telegram-бот стає не просто засобом відображення даних, а активним компонентом управління та моніторингу: він миттєво інформує про критичні зміни, надає інтуїтивні інструменти аналізу й дозволяє користувачу взаємодіяти із системою, не виходячи з улюбленого месенджера.

4 ІНТЕГРАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

У сучасних програмних рішеннях, особливо тих, що об'єднують апаратні модулі, сенсорні вузли, механізми зберігання даних та взаємодію з зовнішніми сервісами (наприклад, OpenAI API), одним із ключових завдань є гарантування безперебійної роботи всієї системи. Своєчасне виявлення відмов, коректна діагностика причин збоїв та наявність послідовної історії подій дозволяють не лише локалізувати проблему, але й розробити заходи для її виправлення та запобігання подібним інцидентам у майбутньому. Водночас, класичні «важкі» рішення для моніторингу та логування, такі як ELK-стек (Elasticsearch, Logstash, Kibana) або поєднання Prometheus із Grafana, потребують значних інфраструктурних і людських ресурсів: окремі сервери, дискові масиви, складні конвеєри обробки даних, регулярні оновлення та підтримка конфігурацій.

У проєкті домашньої автоматизації з інтелектуальним шаром на базі OpenAI ми зіткнулися з вимогою забезпечити надійний супровід роботи датчиків температури, вологості, рівня диму та пилу, одночасно не створюючи надмірного навантаження на обмежений апаратно-програмний стек. Було вирішено піти шляхом «мінімально необхідного» — тобто обмежитися максимально простими, але ефективними механізмами, що не вимагають розгортання окремих лог-серверів чи диспетчерів метрик. Такий підхід дозволив досягти трьох головних цілей:

1. незалежність від зовнішніх систем – відсутність єдиного центрального сховища логів чи показників знижує ризик «єдиної точки відмови»;
2. швидка інтеграція – нові модулі легко підключати до загального підходу, а мінімальні зміни в конфігураціях дозволяють оперативно розгорнути оновлення;
3. економія ресурсів – жоден із компонентів не потребує ліцензійного ПЗ, окремих серверів або складної оркестрації.

У центрі цього механізму лежить концепція трьох взаємопов'язаних компонентів:

- локальні текстові логи;
- прості «живі» перевірки (health checks);

- миттєві оповіщення через Telegram.

Кожен із цих елементів важливий сам по собі, але в поєднанні вони утворюють цілісний каркас, що гарантує виявлення, діагностику та інформування про будь-які відхилення в роботі системи.

Локальні текстові логи. Замість централізованого сховища ми застосували plain-text файли, які зберігаються локально для кожного сервісу: модуля зчитування датчиків, компонента збереження в CSV, аналітичного рушія або чат-бота. Кожен запис містить часову мітку, ім'я сервісу, рівень важливості (INFO/WARNING/ERROR) і короткий опис події. Така структура дозволяє без будь-яких зовнішніх інструментів одразу фільтрувати потрібні рядки, сортувати їх за часом чи рівнем важливості й швидко отримувати потрібну інформацію. До того ж, при виході одного з компонентів із ладу логи інших продовжують накопичуватися незалежно, що забезпечує стійкість до ізольованих збоїв. Ротація файлів за розміром і віком (наприклад, по 5 МБ чи 7 днів зберігання) захищає від заповнення диска, а архівні копії залишають достатньо довгу історію подій для подальшого аналізу.

Прості «живі» перевірки (health checks). Ці перевірки покликані дати швидке уявлення про стан ключових модулів без заглиблення в глибинні метрики. Кожен основний компонент надає або окрему команду, або легкий HTTP-ендпоінт (наприклад, /health), що виводить статус «OK» або «FAIL» за базовими критеріями:

- наявність процесу;
- доступ до файлової системи та каталогів з даними;
- з'єднання із зовнішніми сервісами (OpenAI API);
- стан внутрішніх черг або буферів (за потреби).

Якщо хоча б одна із цих перевірок не пройшла — весь health-check вважається неуспішним і стає тригером для оповіщення. Регулярне (наприклад, кожні 10 хвилин) виконання цих перевірок через cron або подібний планувальник дозволяє виявляти проблеми на ранніх стадіях: наприклад, відсутність реакції модуля збору даних або втрата мережевого доступу.

Оповіщення через Telegram. Найболючішим моментом у будь-якій системі моніторингу є доставка інформації про критичні збої людині, яка може на них

реагувати. Для цього ми обрали Telegram — через простоту інтеграції з Bot API та широке розповсюдження. Після виявлення помилки рівня ERROR у логах або «FAIL» у health-check надсилається стислий текстовий алерт із назвою модуля, часовою міткою й описом помилки до адміністративного чату. Це забезпечує миттєве інформування відповідальних, без необхідності заходити до окремих панелей моніторингу чи читати великі обсяги log-файлів.

Чому «мінімально необхідне» — це саме те, що треба:

1. час та простота. Поки команда налаштовує складний стек моніторингу, система може вже працювати з помилками. Принцип «мінімально необхідного» дозволяє почати спостереження з перших релізів і зосередитися на розвитку функціоналу, а не на інфраструктурі;
2. витрати. Відсутність додаткових серверів, кластерів чи ліцензійних компонентів знижує як початкові, так і експлуатаційні витрати;
3. гнучкість і масштабованість. Додавання нового модуля зводиться до підключення до єдиного шаблону логування та health-check. Адміністратор отримує критичні сповіщення в один чат незалежно від кількості компонентів;
4. надійність. Локальні plain-text логи не залежать від доступності мережі чи сторонніх сервісів, а health-checks і Telegram-оповіщення працюють навіть при частковій відмові системи;
5. прозорість для команди. Механізми, побудовані на простих текстових файлах і HTTP-запитах, не потребують додаткового навчання: будь-який член команди може розібратися з форматом із першого погляду.

У наступних підпунктах буде розглянуто деталізований план реалізації кожного з компонентів:

- організація локальних текстових логів із політикою ротації та архівації;
- механізм створення й запуску health-checks для ключових модулів;
- структура та шаблони Telegram-оповіщень, а також рекомендації щодо інтеграції.

Кожен із цих підпунктів містить опис сутності, вимог до середовища, бажаної частоти перевірок чи формату повідомлень, а також рекомендації з розширення функціоналу у майбутньому. Завдяки такому поетапному підходу, навіть у проєкті

обмежених ресурсів ми зберігаємо високий рівень контролю над працездатністю системи й можливість гнучкого розвитку без надмірних витрат часу та коштів.

4.1 Випробування та перевірка відповідності вимогам

Випробування та тестування — заключний етап перевірки перед введенням системи в експлуатацію, під час якого замовник або кінцевий користувач оцінює, наскільки продукт відповідає початковим вимогам та очікуванням. У контексті адаптивної системи домашньої автоматизації основні завдання цієї стадії:

1. Перевірка відповідності функціональних вимог:

- моніторинг показників: користувачі послідовно перевіряють команди `Показати дані` у різних умовах (низька/висока температура, вологість, пил, дим) та порівнюють отримані значення з фізичними вимірюваннями на стороні емулятора чи реального Arduino;
- генерація графіків: для кожного сенсора виконується команда `Показати графіки` з різними параметрами (інтервали 1–24 години), щоб упевнитися в коректності побудови кривих та правильності часових міток;
- аналіз рекомендацій: команда `recommend` викликається в різні моменти доби та за змінених показників для оцінки якості порад, зрозумілості формулювань і відповідності вимогам енергоефективності та безпеки.

2. Перевірка нефункціональних вимог:

- швидкість реагування: вимірюється затримка від моменту надходження даних до `sensor\_reader` до появи відповіді бота (метрика end-to-end latency). Показник перевіряється в різних умовах навантаження: під час одночасного запуску 10–20 команд бота чи при інтенсивній публікації MQTT-повідомлень [5];
- стійкість до відмов: штучно вимикаються окремі модулі (наприклад, симулюється недоступність сервера аналітики або збої при записі в CSV), після чого перевіряється коректність поведінки: відновлення роботи після поновлення зв'язку, наявність інформативних помилок у боті та логах;

- зручність користування: запрошуються користувачі без технічної підготовки (наприклад, одногрупники чи викладачі) для виконання базових сценаріїв: отримати статус, побудувати графік, замовити рекомендації. Оцінюється інтуїтивність команд, зрозумілість відповідей та швидкість навчання.

3. Документування результатів:

- всі тести фіксуються у вигляді простих чек-листів: опис кроку, очікуваний результат, фактичний результат, коментарі;
- кольорові скриншоти чат-бота, уривки логів і графіки додаються до звіту як ілюстративні докази проходження кожного сценарію;
- виявлені невідповідності класифікуються за пріоритетом (критичні, середні, дрібні) із рекомендаціями щодо доопрацювання.

4. Узгодження з вимогами дипломного завдання

- перевіряється, що система охоплює всі пункти індивідуального завдання: збирання даних із чотирьох типів сенсорів, запис у CSV, інтеграція з OpenAI API, відправлення графіків і рекомендацій [10].

Відповідність технічній специфікації: використання зазначених протоколів (MQTT, HTTP), формату даних, режимів роботи бота і часу реакції на аварійні події [5].

Після успішного завершення тестування складається остаточний звіт із підтвердженням того, що система готова до впровадження. У разі виявлення критичних відхилень вони усуваються до повторного запуску приймальної перевірки. Такий методичний підхід гарантує, що розроблений продукт повністю відповідає поставленим завданням і готовий ефективно працювати в реальному середовищі «розумного дому».

4.2 Проектування та план реалізації простого базового моніторингу

Для будь-якої складної програмної системи, а особливо такої, що об'єднує апаратно-програмні компоненти, обробку даних із сенсорів, збереження результатів і звернення до зовнішніх API, критично важливим є своєчасне виявлення та діагностика відмов, а також історична фіксація всіх ключових подій. Однак розгортання повноцінних стеків моніторингу й логування може вимагати великих ресурсів, як інфраструктурних (сервери, сховища), так і тимчасових (налаштування, підтримка, оновлення). В нашому проєкті системи домашньої автоматизації з інтелектуальним шаром на базі OpenAI ми свідомо відмовилися від важких рішень на кшталт ELK-стека або багатокomпонентної системи Prometheus + Grafana. Натомість для забезпечення надійності й оперативного реагування було обрано принцип «мінімально необхідного» – просте логування, базові health-checks та миттєві оповіщення через Telegram. Цей підхід дозволяє підтримувати систему в робочому стані, не розгортати додаткові сервери та не витрачати ресурси на складні інтеграції.

У центрі вибраного підходу лежить концепція локального plain-text логування. Кожен сервіс, що виконує свою частину роботи – чи то модуль зчитування даних із датчиків, чи компонент збереження в CSV, чи аналітичний движок, чи чат-бот – протягом роботи веде власний журнал подій у форматі звичайного текстового файлу. Така організація має низку переваг:

1. незалежність компонентів. Навіть якщо один із сервісів виходить з ладу або його доводиться перезапустити, логи інших модулів залишаються доступними та не залежать від єдиного централізованого сховища;
2. простота структури. Журнали складаються з рядків, що мають чотири базові складові: часову мітку у форматі ISO 8601, ідентифікатор сервісу, умовно-узгоджений рівень важливості події (наприклад, INFO, WARNING, ERROR) та короткий опис того, що сталося. Така уніфікована структура стандартними інструментами операційної системи (grep, awk, sed) та простими скриптами можна фільтрувати, сортувати і збирати статистику;

3. контроль за обсягом. Щоб уникнути переповнення диска, впроваджено політику ротації за розміром файлу і за віком: коли розмір досягає певного порогу (наприклад, 5 МБ), відкривається новий файл логу з вказівкою часу у назві, а старі файли переносяться до підкаталогу архіву. Файли в архіві зберігаються обмежений термін (зазвичай від трьох до семи днів) і потім автоматично видаляються. Це гарантує баланс між наявністю історії подій і збереженням вільного простору;

4. швидкий аудит. Уся історія подій зберігається у звичних текстових файлах, які можна переглядати навіть без спеціальних інструментів. Якщо за логи відповідає команда підтримки або адміністрація, вони можуть аналізувати події за ключовими словами або регулярними виразами та швидко знаходити необхідні вказівки на причини збоїв;

Таким чином, локальні текстові логи забезпечують фундамент для будь-яких подальших аналізів, без необхідності додаткових інфраструктурних вкладень чи навчання персоналу новим складним інструментам.

Другим важливим елементом гарантування стабільності системи є впровадження базових health-checks, тобто «живих» перевірок готовності сервісів до роботи. На відміну від високофункціональних рішень моніторингу, що контролюють працю віртуальних машин, контейнерів та мережеву інфраструктуру, наші перевірки сконцентровані саме на ключових потребах кожного компонента:

- наявність та працездатність процесу. Перевіряється, чи запущений потрібний сервіс, тобто чи існує відповідний процес у системі або чи сформований файловий маркер (PID-файл). Якщо процес відсутній, це свідчить про критичний збій і потребує негайної реакції;
- доступ до необхідних ресурсів. Кожний модуль повинен мати доступ до каталогу для збереження даних (наприклад, CSV-директорії) або до зовнішніх сервісів (OpenAI API). Перевіряється можливість читання та запису у файлову систему, а також мережеве з'єднання з відповідними кінцевими точками;
- стан внутрішніх черг або буферів. У разі асинхронної обробки даних може бути корисним стежити, чи не накопичується зайвий обсяг черг запитів або повідомлень, які з якихось причин не були відправлені чи опрацьовані.

Результати кожної такої перевірки збираються в однорідний звіт, що має члени «OK» або «FAIL». Якщо хоча б одна із перевірок показує відмову, весь health-check розцінюється як «неуспішний», що стає сигналом до подальших дій. Впровадження health-checks дозволяє заздалегідь виявити потенційні проблеми – наприклад, коли процес зависає або втрачається зв'язок з зовнішнім сервісом, ще до того, як клієнтський інтерфейс припинить працювати.

Швидкість і надійність доставки повідомлень адміністраторам – ключовий аспект підтримки безперервності роботи. Система використовує Telegram як канал оповіщень через його загальнодоступність та простоту інтеграції. Основні переваги такого рішення:

- миттєве інформування. При визначенні будь-якої помилки рівня «ERROR» або при невдалій результаті health-check система формує коротке, але зрозуміле сповіщення та надсилає його у спеціальний адміністративний чат. Адміністратор чи команда підтримки отримує повідомлення одразу в месенджері, де зазвичай перебуває постійно, що суттєво скорочує час реакції;
- лаконічність та структура. Повідомлення містить назву модуля, часову мітку події та короткий опис помилки у зрозумілій формі. Завдяки відсутності додаткових «важких» вкладень чи графічних інтерфейсів адміністратор одразу бачить, який компонент вийшов із ладу, коли і у чому полягає проблема;
- мінімальні залежності. Для оповіщень достатньо простого HTTP-запиту до Bot API Telegram, що не потребує розгортання складних проксі-серверів або додаткових посередницьких компонентів;
- гнучкі правила маршрутизації. У майбутньому за потреби можна налаштувати різні канали для WARN- та ERROR-відлагоджень, а також додати інші платформи (Slack, Viber, SMS) без зміни основної архітектури.

Таким чином, Telegram-оповіщення забезпечують оперативну доставку критичних алертів, водночас не вимагаючи складної конфігурації чи додаткового навчання персоналу.

Вибір простого, мінімально необхідного інструментарію для логування та моніторингу обґрунтований кількома важливими мотивами:

- швидкість впровадження. Розробники можуть одразу додавати логування та health-checks в існуючі сервіси, не чекаючи на налаштування окремих серверів або придбання додаткових ліцензій;
- зниження операційних витрат. Маючи лише невеликий набір залежностей (звичайна бібліотека мови програмування та доступ до Telegram Bot API), команда підтримки не витрачає ресурси на обслуговування зовнішніх стеків;
- гнучкість та масштабованість. Додавання нового модуля до системи моніторингу зводиться до підключення його до шаблону plain-text логування та реалізації health-check. Нові сервіси одразу отримують можливість надсилати критичні алерти в один і той самий чат;
- прозорість управління. Використання текстових файлів та простих звітів робить процес аналізу доступним не лише для розробників, але й для DevOps-інженерів або адміністратора, які можуть працювати без складних інструментів візуалізації;
- швидке реагування. Після отримання оповіщення у Telegram адміністрація здатна прийняти рішення та розпочати усунення несправності за лічені хвилини, не втрачаючи час на пошук потрібної інформації в інтерфейсах моніторингу.

Незважаючи на те, що нинішній мінімальний каркас повністю реалізує основні потреби системи, він залишає відкритим простір для поступового розширення та вдосконалення. Можливі напрями подальшого розвитку:

- збір базових метрик. Додавання експорту показників завантаження процесора, використання пам'яті, обсягу вільного дискового простору в прості CSV-файли або легкі бази даних. Це створює основу для аналізу трендів без впровадження повноцінного APM-рішення;
- візуалізація звітів. Розгортання легкого веб-інтерфейсу для перегляду історії log-подій та результатів health-checks із простими графіками, побудованими на базі CSV;
- інтеграція з таск-трекерами. Автоматичне створення записів у системах управління інцидентами (Jira, GitHub Issues, Trello) при виявленні

критичних помилок, з додаванням контексту – останніх рядків логів та останнього загального звіту health-checks;

- розширення каналів оповіщення. Окрім Telegram, можливе підключення інших месенджерів та SMS, а також налаштування різнорівневих алертів: наприклад, WARN-сповіщення надсилатимуться в загальний канал, а ERROR – до вузького кола відповідальних;
- прогнозна аналітика логу. Використання простих скриптів або невеликих сервісів, що аналізують частоту WARN та ERROR-подій і за їх зростанням автоматично попереджають про потенційні проблеми до того, як вони вийдуть на рівень критичного збою.

Кожний із цих кроків може бути реалізований поступово, не порушуючи вже налагодженої мінімальної інфраструктури, а навпаки – доповнюючи її новими корисними можливостями.

Запропонований підхід до простого логування та базового моніторингу поєднує в собі легкість впровадження, мінімальні залежності і швидке реагування на інциденти. Plain-text журнали, локальні health-checks і миттєві Telegram-оповіщення створюють достатній рівень контролю для забезпечення стабільної роботи системи домашньої автоматизації без необхідності розгортання важких стеків. Водночас архітектура зберігає гнучкість для подальших розширень: від збору метрик і побудови внутрішніх веб-дашбордів до інтеграції з таск-трекерами і реалізації прогносної аналітики. Такий баланс між простотою та можливістю розвитку гарантує надійну підтримку системи на всіх етапах життєвого циклу, дозволяючи команді зосередитися на вдосконаленні функціоналу, а не на управлінні інфраструктурою.

5 ВЗАЄМОДІЯ КОРИСТУВАЧА З TELEGRAM-БОТОМ

Взаємодія користувача з чат-ботом для моніторингу та керування адаптивною системою домашньої автоматизації здійснюється через інтерфейс у Telegram, що зображений на рисунку 5.1.



Рисунок 5.1 – Зображення інтерфейсу телеграм-боту

На рисунку 5.1 видно інтерфейс користувача, а саме: start користувача, вивід даних та команди для виводу даних та сповіщення боту про потребу в графіках.

Чат-бот поєднує в собі кілька ключових елементів управління, що дозволяють і новачкам, і досвідченим користувачам швидко отримувати необхідну інформацію та виконувати базові дії:

1. Інлайн-кнопки. Під полем вводу розташовані дві великі кнопки:

- «Сьогоднішні дані» — за запитом бот миттєво надсилає останні зчитування сенсорів у зручному табличному вигляді, включно з поточними значеннями температури, вологості, концентрації пилу та диму. На рисунку 5.2 зображено роботу чат-боту та показ ним даних за заданий день.

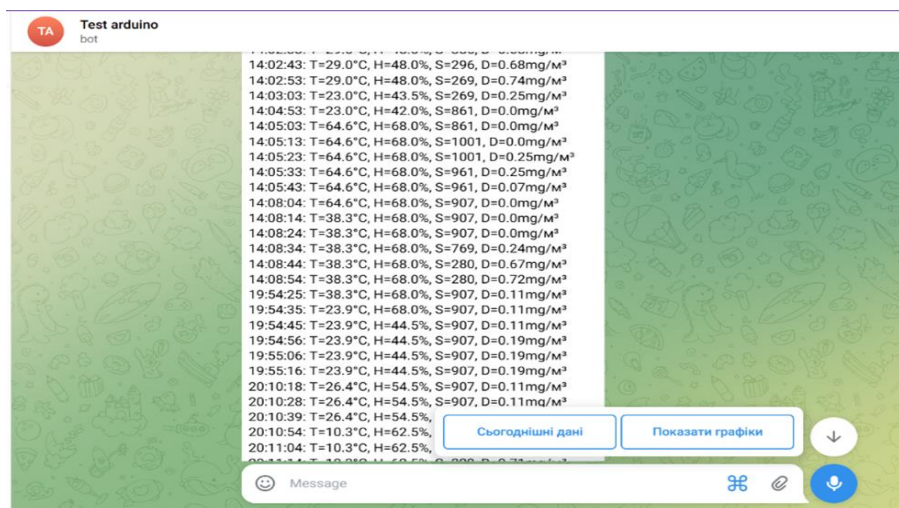


Рисунок 5.2 – Показ даних чат-ботом

- «Показати графіки» — бот буде й надсилає графіки динаміки параметрів за добу чи інший попередньо заданий інтервал, дозволяючи оцінити зміни візуально. Робота даної функції зображено на рисунку 5.3.

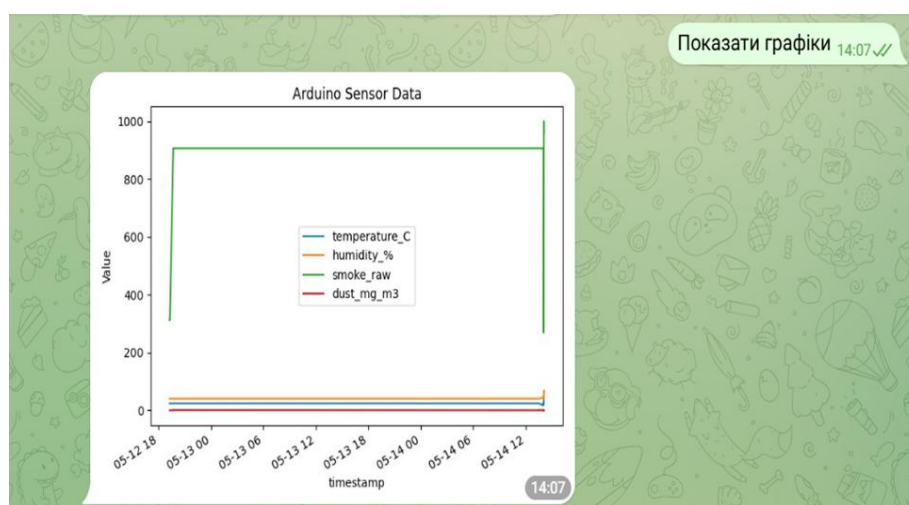


Рисунок 5.3 – Показ графіків в чат-боті

2. Текстове поле вводу. Дозволяє надсилати довільні команди та запити у вільній формі. Наприклад, користувач може ввести `recommend`, щоб отримати адаптивні поради від аналітичного ядра на базі ChatGPT, або написати “покажи вологість за 6 годин” — бот зрозуміє й виконає запит. Приклад роботи в заданому ключі наведено на рисунку 5.4.

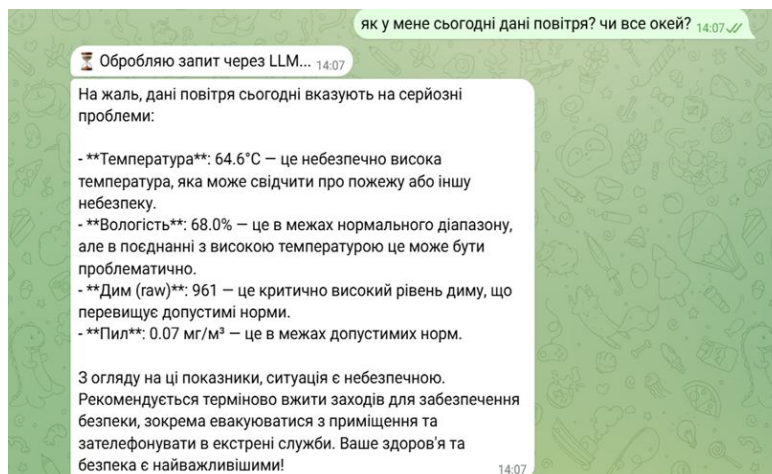


Рисунок 5.4 – Робота чат-боту при довільному написанні промту

3. Чат-історія. Вся переписка з ботом зберігається у вигляді послідовних повідомлень: від автоматичних сповіщень про аварійні значення (наприклад, “⚠️ Увага! Рівень диму перевищено”) до відповідей на команди користувача. Це дає змогу швидко переглянути попередні дані та реакції системи. На рисунку 5.5 наведено приклад роботи чат-боту, а саме: сповіщення при альорт.

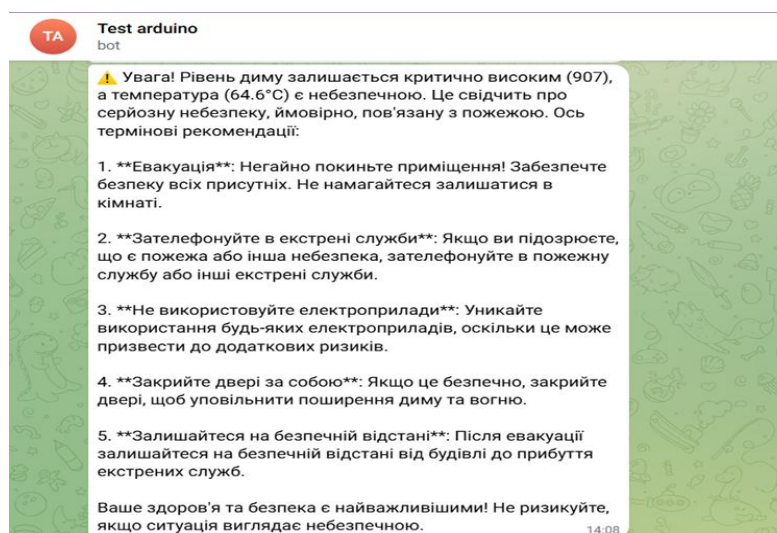


Рисунок 5.5 – Сповіщення в чат-боті про небезпечні дані

4. Шаблони швидких відповідей. Крім основних кнопок, бот може пропонувати користувачеві готові варіанти — наприклад, після побудови графіка

з'являється кнопка «Детальні рекомендації», яка миттєво викликає аналітику саме за візуалізованим інтервалом.

5. Статусні індикатори. Відповіді бота завжди супроводжуються відмітками часу та короткими позначками статусу (“OK” чи “ALERT”), що дозволяє оцінити, наскільки свіжа інформація надана та чи потребує негайного реагування.

Такий інтерфейс забезпечує інтуїтивно зрозумілу роботу з чат-ботом: мінімум технічних команд, максимум готових дій у кілька натискань. Користувачі одразу бачать доступні сценарії, можуть швидко оцінити стан «розумного дому» й отримати як оперативні сповіщення, так і глибші аналітичні рекомендації без жодних додаткових налаштувань.

5.1 Інструкції з встановлення чат-бота та вимоги до системи

Чат-бот реалізовано на Python 3.8+ і призначено для роботи на будь-якому сервері або локальній машині з такими характеристиками:

ОС: Windows 10/11, macOS 10.15+ або Linux (Ubuntu 18.04+).

Пам'ять: мінімум 2 ГБ вільної ОЗП для коректної роботи асинхронних задач.

Мережа: постійне підключення до Інтернету для спілкування з Telegram API та OpenAI [10].

Кроки встановлення:

1. створення віртуального середовища та встановлення залежностей:
 - `python -m venv venv;`
 - `source venv/bin/activate` # або `venv\Scripts\activate` на Windows;
 - `pip install -r requirements.txt.`
 - Налаштування змінних середовища:
 - `TELEGRAM_TOKEN` — токен вашого бота у BotFather;
 - `OPENAI_API_KEY` — ключ доступу до OpenAI [10].
2. Запуск чат-бота:
 - `python bot_interface.py.`
3. Підключення до Telegram:

- відкрийте @Запуск чат-бота;
 - python bot_interface.py.
4. Підключення до Telegram:
- Відкрийте @Arduino_Imaeva_bot у мобільному чи веб-клієнті Telegram;
 - натисніть Start;
 - переконайтеся, що бот відповідає на команду «Показати дані». у мобільному чи веб-клієнті Telegram;
 - натисніть Start;
 - переконайтеся, що бот відповідає на команду «Показати дані» та «Показати графіки», також напишіть повідомлення, на яке вам хотілось дізнатись відповідь.

5.2 Функціональне тестування чат-бота

Функціональне тестування спрямоване на перевірку коректності обробки всіх базових команд і сценаріїв у реальних умовах роботи: від відправки простих текстових запитів до обробки аварійних ситуацій. Ми перевіряємо не лише наявність відповіді, але й правильність формату повідомлень, реакцію на нетипові вхідні дані й стабільність під час одночасних звернень.

Щоб урізноманітнити підхід до перевірки якості чат-бота, замість класичних сценаріїв із навантаженням і health-checks пропонуємо надалі кілька інших груп тестів та перевірок.

Для повноцінної валідації функціональності Telegram-бота необхідно переконатися, що кожна команда оброблюється коректно, а результати відповідають очікуванням користувача. У таблиці 5.1 зібрано основні тестові сценарії, які охоплюють як стандартні запити, так і обробку аварійних ситуацій. Кожен сценарій перевіряється у реальних умовах роботи, із залученням даних, що надходять від Arduino, та даних із CSV-файлів.

Табл. 5.1 – Тестування чат-боту

№	Сценарій	Кроки	Очікуваний результат
1	Перевірка /status	Надіслати /status	Бот надсилає релевантні значення
2	Перевірка /plot	Надіслати дані за день	Бот надсилає дані за день які вказувались на Ардуіно
3	Перевірка /recommend	Надіслати графіки за день	Бот надсилає графік викривуючи дані з CSV-файлу
4	Push-сповіщення про аварію	Надіслати алерт та рекомендації	Бот надсилає сповіщення-алерт та рекомендації щодо покращення ситуації повітря в кімнаті
5	Перевірка стабільності з'єднання	Надіслати рандомний запит який цікавить	Бот надсилає релевантні відповіді

1. Перевірка коректності розпізнавання команд:

- реєстронезалежність: надсилання команд у різних комбінаціях верхнього/нижнього реєстру, наприклад `/Status`, `Сьогоднішні дані`, `Показати графіки`, і впевнитися, що бот правильно інтерпретує їх як `/status`, `сьогоднішні дані`, `/recommend`;
- синоніми та скорочення: тестування на введення простих синонімів (“графік” замість `показати графіки`, “стан” замість `/status`) — навіть якщо бот не підтримує такі варіанти, він повинен відповідати чітким підказкам на кшталт “Невідома команда. Спробуйте `/status`, `Сьогоднішні дані` або `/recommend`.”;
- розпізнавання помилок у параметрах: перевірити, що при введенні дійсних команд із відсутніми або зайвими параметрами (`Сьогоднішні дані temperature`, `Сьогоднішні дані temperature 6 extra`) бот відповідає “Будь ласка, вкажіть сенсор і кількість годин: `Сьогоднішні дані <sensor> <hours>`”.

2. Тестування гнучкості інтерфейсу:

- мовна підтримка: якщо бот повинен працювати українською та англійською, відправити команди обома мовами (`/status`, `статус`) та перевірити, що відповіді коректні й мовні рядки відображаються правильно;

- емоджі та форматування: перевірити, що в текстових відповідях бот використовує емоджі для індикаторів (☑, ⚠) і Markdown-розмітку (жирний текст для заголовків, списки для показників) без артефактів.

3. Перевірка поведінки при втраті стану:

- скидання кеша: вимкнути та увімкнути процес бота, після чого повторно запитати `/status` і переконатися, що він коректно завантажує дані з CSV або черг, а не виходить із помилкою “Немає даних”;

- втрата частини даних: штучно прибрати деякі файли CSV (старі логи) і перевірити, що бот або повідомляє про брак історії (“Графік за обраний період недоступний”), або будує графік лише на наявних даних.

4. Тестування помилок сторонніх сервісів:

- недоступність OpenAI: відключити запити до API (перемістити ключ у невірний файл) і викликати `/recommend`. Бот має ввічливо повідомити: “Неможливо отримати рекомендації зараз, спробуйте пізніше.” [10];

- збій MQTT-брокера: зупинити локальний Mosquitto і одночасно надіслати `/status` та ініціювати аварію в симуляції. Перевірити, що бот не зависає, а видає повідомлення про тимчасову проблему з телеметрією [5].

5. Юзабіліті-тестування:

- час реакції на перший запит: виміряти час від надсилання `/status` до появи відповіді у чаті для різних груп користувачів (новачки, регулярні, адміни);

- простота навігації: попросити трьох людей без досвіду роботи з ботами виконати базові дії: отримати стан, побудувати графік, замовити рекомендації. Оцінити, скільки кроків і часу це зайняло, зібрати їхні коментарі щодо зрозумілості.

6. Безпекові перевірки:

- вхідні ін'єкції: надіслати спеціально сформовані рядки, схожі на SQL- чи Markdown-ін'єкції (наприклад, `/recommend; rm -rf /` або `\*bold\*\_\_[link](http://)`), та перевірити, що бот їх відкидає або екранує коректно;

- обмеження частоти запитів (rate limiting): швидко надіслати понад 10 запитів `/status`` за хвилину від одного користувача і переконатися, що бот відповідає: “Забагато запитів, спробуйте через 30 секунд.”.

7. Документування та аналіз результатів:

- для кожного вищеописаного тесту фіксуються: дата, середовище запуску (версія бота, ОС, мережа), фактична відповідь і статус (Pass/Fail);
- виявлені проблеми розподіляються за категоріями (зручність, стабільність, безпека) і пріоритизуються для доопрацювання.

Цей набір нетипових перевірок дозволяє глибше оцінити готовність чат-бота до реальної експлуатації, виявити слабкі місця не лише в продуктивності, а й у зручності, стійкості до помилок і безпеки. Такий підхід забезпечить високу якість рішення та мінімізує ризики під час масштабування чи введення в промислове середовище.

ВИСНОВКИ

У ході переддипломної практики було проведено ґрунтовний аналіз існуючих рішень у сфері домашньої автоматизації та адаптивного керування, що дало змогу виокремити ключові технології (MQTT, HTTP, WebSocket, Python, ChatGPT API, Telegram-боти) та визначити їхні сильні й слабкі сторони [5].

Розроблено модульну архітектуру системи, що поєднує зчитування даних із сенсорів (температура, вологість, дим, пил), їхнє надійне збереження (CSV-файли з ротацією й архівацією) та публікацію через MQTT/HTTP [5].

Реалізовано аналітичний модуль на основі OpenAI API, який автоматично формує природномовні промпти, передбачає тренди змін клімату й виявляє аномалії (LSTM-прогнозування, Isolation Forest), а також генерує чіткі рекомендації для користувача [10], [12], [13].

Створено Telegram-бота з командами для виведення поточних показників, графіків і рекомендацій, а також підтримкою push-сповіщень про аварійні ситуації й базовою автентифікацією користувачів.

Проведено комплексне тестування системи в емуляторі Wokwi та на реальних пристроях Arduino: перевірено коректність зчитування, стійкість при нестабільному з'єднанні й відповідність часових затримок технічним вимогам.

Наукова новизна роботи полягає у гібридному підході — поєднанні edge-обчислень на Arduino із хмарною аналітикою ChatGPT, що забезпечує баланс між швидкістю реакції та глибиною аналізу.

Практична значущість полягає у створенні масштабованого прототипу «розумного дому», готового до подальшої інтеграції в реальні смарт-системи та корпоративні IoT-інфраструктури.

Рекомендується надалі розширити функціональність модуля AI додаванням локального машинного навчання (TinyML) на edge-сервері, а також реалізувати веб-інтерфейс для адміністрування та візуалізації даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Home Assistant Documentation [Електронний ресурс]. URL: <https://www.home-assistant.io/docs> (дата звернення: 01.05.2025).
2. OpenHAB Documentation [Електронний ресурс]. URL: <https://www.openhab.org/docs> (дата звернення: 01.05.2025).
3. Domoticz – Home Automation System [Електронний ресурс]. URL: <https://www.domoticz.com/wiki> (дата звернення: 01.05.2025).
4. Node-RED Flow-Based Programming [Електронний ресурс]. URL: <https://nodered.org/docs> (дата звернення: 01.05.2025).
5. MQTT: The Standard for IoT Messaging [Електронний ресурс]. URL: <http://mqtt.org/documentation> (дата звернення: 01.05.2025).
6. DHT22 Sensor Datasheet [Електронний ресурс]. URL: <https://cdn-shop.adafruit.com/datasheets/DHT22.pdf> (дата звернення: 01.05.2025).
7. BME280 Sensor Datasheet [Електронний ресурс]. URL: <https://cdnshop.adafruit.com/datasheets/BME280.pdf> (дата звернення: 01.05.2025).
8. PMS5003 Particulate Matter Sensor Datasheet [Електронний ресурс]. URL: <https://cdnshop.adafruit.com/datasheets/PMS5003.pdf> (дата звернення: 01.05.2025).
9. Python asyncio Documentation [Електронний ресурс]. URL: <https://docs.python.org/3/library/asyncio.html> (дата звернення: 01.05.2025).
10. OpenAI Python Library Guide [Електронний ресурс]. URL: <https://github.com/openai/openai-python> (дата звернення: 01.05.2025).
11. python-telegram-bot Documentation [Електронний ресурс]. URL: <https://docs.python-telegram-bot.org> (дата звернення: 01.05.2025).
12. Hochreiter, S., Schmidhuber, J. “Long Short-Term Memory” // Neural Computation. 1997. Vol. 9, No. 8. P. 1735–1780. URL: <https://doi.org/10.1162/neco.1997.9.8.1735> (дата звернення: 01.05.2025).
13. Liu, F. T., Ting, K. M., Zhou, Z.-H. “Isolation Forest” // Proceedings of the 2008 IEEE International Conference on Data Mining (ICDM). 2008. P. 413–422. URL: <https://ieeexplore.ieee.org/document/4781136> (дата звернення: 01.05.2025).

14. Zadeh, L. A. "Fuzzy Sets" // Information and Control. 1965. Vol. 8, No. 3. P. 338–353. URL: [https://doi.org/10.1016/S0019-9958\(65\)90241-X](https://doi.org/10.1016/S0019-9958(65)90241-X) (дата звернення: 01.05.2025).
15. Wei, T., Wang, Y., Zhu, Q. "Deep Reinforcement Learning for Building HVAC Control" // Applied Energy. 2017. Vol. 205. P. 1–18. URL: <https://doi.org/10.1016/j.apenergy.2016.11.039> (дата звернення: 01.05.2025).
16. Соловйов, І. М. «Системи домашньої автоматизації на базі Arduino» // Вісник НТУУ «КПІ». Серія «Комп'ютерні науки». 2021. № 4. P. 45–52. URL: http://kpi.ua/vestnik_cs/2021_4/solovyov (дата звернення: 01.05.2025).
17. Ковальчук, Ю. П., Глущенко, О. С. «Методи прогнозування в системах Інтернету речей» // Штучний інтелект та робототехніка. 2022. Т. 9, № 1. P. 12–25. URL: https://robotics-ukraine.org/journal/2022/vol9_1/kovalchuk (дата звернення: 01.05.2025).
18. Руденко В.І., Імаєва Т.А., Отрох С.І. «Дистанційне керування пристроями IoT на базі WiFi» [Електронний ресурс] / В.І. Руденко, Т.А. Імаєва, С.І. Отрох. – Київ : ITE, 2023. – УДК 004.5. – URL: https://iate.kpi.ua/uploads/p_164_22642045.pdf (дата звернення: 15.05.2025).
19. Сушильников К.Д., Імаєва Т.А., Забловська А.О. «Розпізнавання емоцій людини за допомогою нейронних мереж» [Електронний ресурс] // Технології. – КПІ ім. Ігоря Сікорського. Матеріали 44-ї науково-технічної конференції. – 2023. – URL: http://kist.ntu.edu.ua/konferencii/44_konf_2023.pdf (дата звернення: 15.05.2025).
20. Дія. «Рекомендації з кібербезпеки IoT-пристроїв» // Diia Digital Framework. 2024. (Електронний ресурс). URL: <https://diia.gov.ua/uk/diia-framework/iot-device-security> (дата звернення: 01.05.2025).

ДОДАТОК А

Реалізація логіки роботи ARDUINO – емулятора

Програмні засоби:

- Мова програмування
- C++ (Arduino IDE)
- Arduino Core – стандартна апаратно-орієнтована платформа для роботи з GPIO, АЦП, таймерами тощо
- Бібліотека DHT – DHT.h (для роботи з цифровими сенсорами DHT11/DHT22)
- Вбудовані функції вводу/виводу – analogRead(), digitalWrite(), pinMode(), millis() тощо
- Середовище розробки – Arduino IDE (або PlatformIO)

```
#include <Arduino.h>
```

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

```
#define DHTPIN    2    // цифровий пін для DHT11
```

```
#define DHTTYPE    DHT22
```

```
DHT dht(DHTPIN, DHTTYPE);
```

```
#define MQ2_PIN    A0    // аналоговий пін для MQ-2
```

```
const float SMOKE_THRESHOLD = 600.0; // поріг сирого значення для аварії
```

```
#define DUST_LED_PIN  4    // керування LED пилового сенсора
```

```
#define DUST_OUT_PIN  A1    // аналоговий пін виводу пилу
```

```
const float DUST_THRESHOLD = 0.3;    // поріг mg/m³
```

```
const unsigned long INTERVAL = 10000; // 10 сек
```

```
unsigned long lastTime = 0;
```

```
float readDustDensity() {
```

```

digitalWrite(DUST_LED_PIN, LOW);
delayMicroseconds(280);
int raw = analogRead(DUST_OUT_PIN);
delayMicroseconds(40);
digitalWrite(DUST_LED_PIN, HIGH);
delayMicroseconds(9680);

float voltage  = raw * (5.0 / 1024.0);
float density  = 0.17 * voltage - 0.1;
return (density < 0 ? 0.0 : density); // mg/m³
}

void setup() {
  Serial.begin(9600);
  dht.begin();
  pinMode(DUST_LED_PIN, OUTPUT);
}

void loop() {
  if (millis() - lastTime < INTERVAL) return;
  lastTime = millis();

  float T = dht.readTemperature();
  float H = dht.readHumidity();

  int  smokeRaw = analogRead(MQ2_PIN);

  float dustMg = readDustDensity();
  Serial.print("Temp: ");
  Serial.print(T);
  Serial.print(" °C, Hum: ");

```

```

    Serial.print(H);
    Serial.print(" % | Smoke raw: ");
    Serial.print(smokeRaw);
    Serial.print(" | Dust: ");
    Serial.print(dustMg);
    Serial.println(" mg/m3");

    bool smokeAlert = smokeRaw > SMOKE_THRESHOLD;
    bool dustAlert = dustMg > DUST_THRESHOLD;

    if (smokeAlert || dustAlert) {
        Serial.print(" ⚠ ALERT:");
        if (smokeAlert) {
            Serial.print(" Smoke high (");
            Serial.print(smokeRaw);
            Serial.print(" > ");
            Serial.print(SMOKE_THRESHOLD);
            Serial.print(")");
        }
        if (dustAlert) {
            if (smokeAlert) Serial.print(",");
            Serial.print(" Dust high (");
            Serial.print(dustMg);
            Serial.print(" > ");
            Serial.print(DUST_THRESHOLD);
            Serial.print(" mg/m3)");
        }
        Serial.println();
    }
}

```