НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету Інформаційних технологій	**Завідувач кафедри** Комп'ютерних систем, мереж та кібербезпеки
_____Ігор БОЛБОТ (підпис)	_____Дмитро КАСАТКІН (підпис)
«___»_____2026 р.	«___»_____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Розробка системи контролю мікроклімату у вантажних відсіках транспортних засобів з використанням IoT-технологій з інтеграцією Telegram-бота та вебдодатка»

Спеціальність F7 «Комп’ютерна інженерія »

Освітньо-професійна програма «Комп’ютерна інженерія »

Гарант освітньої програми канд. фіз.-мат. наук, доцент	_____ (підпис)	**Євгеній НІКІТЕНКО**
Керівник бакалаврської кваліфікаційної роботи канд. пед. наук, доцент	_____ (підпис)	**Дмитро КАСАТКІН**
Виконав	_____ (підпис)	**Володимир ТОЛКАЧ**

КИЇВ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
канд. пед. наук, доцент_____Дмитро КАСАТКІН
«_____»_____20____ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

ЗДОБУВАЧУ

Толкачу Володимиру Георгійовичу
(прізвище, ім'я, по батькові)

Спеціальність «Комп'ютерна інженерія »

Освітньо-професійна програма «Комп'ютерна інженерія »

Тема кваліфікаційної бакалаврської роботи
«Розробка системи контролю мікроклімату у вантажних відсіках транспортних засобів з використанням IoT-технологій з інтеграцією Telegram-бота та вебдодатка»

затверджена наказом ректора НУБіП України від «10»___12_____2025 р. №___3072 «С»___

Термін подання завершеної роботи на кафедру «2»___06_____2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи

Перелік питань, що підлягають дослідженню:

Перелік графічного матеріалу (за необхідності).

Дата видачі завдання "__10__"_____12_____2025 р.

Керівник бакалаврської кваліфікаційної роботи канд. пед. наук, доцент	_____ (підпис)	**Дмитро КАСАТКІН**
Завдання взяв до виконання	_____ (підпис)	**Володимир ТОЛКАЧ**

КАЛЕНДАРНИЙ ПЛАН

Виконання бакалаврської кваліфікаційної роботи

Студент: Толкач Володимир Георгійович
Група: КІ22012б
Спеціальність: 123 «Комп’ютерна інженерія»
Тема роботи:
«Розробка системи контролю мікроклімату у вантажних відсіках транспортних засобів з використанням IoT-технологій з інтеграцією Telegram-бота та вебдодатка»

№	Етап виконання роботи	Термін виконання	Відмітка
1	Аналіз предметної області та постановка задачі	01.02.2026 – 10.02.2026	Виконано
2	Вибір елементної бази та розробка структурної схеми системи	11.02.2026 – 25.02.2026	Виконано
3	Розробка принципової електричної схеми та підключення модулів	26.02.2026 – 15.03.2026	Виконано
4	Розробка програмного забезпечення системи моніторингу	16.03.2026 – 10.04.2026	Виконано
5	Розробка вебпанелі та інтеграція Telegram-бота	11.04.2026 – 25.04.2026	Виконано
6	Проведення експериментальних досліджень та тестування	26.04.2026 – 05.05.2026	Виконано
7	Аналіз результатів та техніко-економічне обґрунтування	06.05.2026 – 10.05.2026	Виконано
8	Оформлення пояснювальної записки та підготовка до захисту	11.05.2026 – 18.05.2026	Виконано

Студент _____ / Толкач В.Г. /

Керівник роботи _____ / Касаткін Д.Ю. /

РЕФЕРАТ

Пояснювальна записка до дипломної роботи: 100 сторінок, 27 рисунків, 15 таблиць,15 джерел за переліком посилань.

Об’єкт дослідження — процес дистанційного моніторингу параметрів мікроклімату та контролю стану вантажу під час транспортування.

Предмет дослідження — апаратно-програмні засоби побудови інтелектуальної IoT-системи на базі мікроконтролера ESP32 для автоматизації моніторингу параметрів середовища у вантажних відсіках транспортних засобів.

Мета роботи — розробка інтелектуальної системи контролю мікроклімату вантажного відсіку транспортного засобу з функціями автономного логування даних, дистанційного моніторингу та автоматичного сповіщення користувача про критичні зміни параметрів середовища.

У процесі виконання дипломної роботи проведено аналіз сучасних систем моніторингу умов транспортування вантажів, виконано обґрунтування вибору елементної бази та розроблено апаратно-програмний комплекс на основі мікроконтролера ESP32 із підтримкою технологій Інтернету речей (IoT).

У роботі реалізовано систему збору та обробки даних із цифрових та аналогових сенсорів, зокрема датчиків температури і вологості SHT31 та DS18B20, датчика контролю якості повітря MQ135, цифрового люксметра BH1750 і датчика вібрації SW-420. Передача та обробка інформації здійснюється за допомогою інтерфейсів I2C, OneWire та SPI.

Програмна частина системи забезпечує локальне відображення інформації на OLED-дисплеї, автоматичне логування результатів вимірювань на microSD-карту у форматі CSV, створення захищеної вебпанелі моніторингу та інтеграцію з Telegram-ботом для дистанційного контролю параметрів середовища і надсилання аварійних повідомлень.

У дипломній роботі розроблено алгоритми автоматичного підключення до мережі Wi-Fi, обробки сенсорних даних, реєстрації подій та передачі телеметрії через

мережу Інтернет. Проведено експериментальні дослідження точності вимірювань, швидкості передачі повідомлень та стабільності функціонування системи в умовах, наближених до реальної експлуатації.

Результати дослідження підтвердили працездатність розробленого прототипу та його ефективність для використання у логістичних процесах, пов'язаних із перевезенням чутливих до умов навколишнього середовища вантажів.

Ключові слова: ESP32, МОНІТОРИНГ МІКРОКЛІМАТУ, ІНТЕРНЕТ РЕЧЕЙ (IoT), TELEGRAM-БОТ, ВЕБПАНЕЛЬ, СЕНСОРНА МЕРЕЖА, MQ135, SHT31, АВТОМАТИЗАЦІЯ ЛОГІСТИКИ, МОНІТОРИНГ ВАНТАЖІВ.

ABSTRACT

Diploma thesis explanatory note: 100 pages, 27 figures 15 tables, 15 references.

The object of research is the process of remote monitoring of microclimate parameters and cargo condition during transportation.

The subject of research is the hardware and software tools for developing an intelligent IoT-based monitoring system using the ESP32 microcontroller for automated environmental control inside cargo compartments.

The purpose of the work is to develop an intelligent cargo compartment microclimate monitoring system with autonomous data logging, remote monitoring capabilities, and automatic user notifications in case of critical environmental changes.

During the diploma project, an analysis of modern cargo transportation monitoring systems was carried out, the hardware platform was substantiated, and a complete hardware-software complex based on the ESP32 microcontroller with Internet of Things (IoT) support was developed.

The implemented system provides data acquisition and processing from digital and analog sensors, including SHT31 and DS18B20 temperature and humidity sensors, MQ135 air quality sensor, BH1750 digital light sensor, and SW-420 vibration sensor. Data exchange and processing are performed using I2C, OneWire, and SPI interfaces.

The software subsystem provides real-time data visualization on an OLED display, autonomous logging of sensor values to a microSD card in CSV format, a secure web dashboard for monitoring, and Telegram bot integration for remote control and emergency notifications.

The diploma thesis includes the development of Wi-Fi auto-configuration algorithms, sensor data processing methods, event logging mechanisms, and Internet-based telemetry transmission. Experimental studies of measurement accuracy, message delivery speed, and overall system stability were conducted under conditions close to real operation scenarios.

The obtained results confirmed the functionality and effectiveness of the developed prototype for use in logistics processes related to transportation of environmentally sensitive cargo.

Keywords: ESP32, MICROCLIMATE MONITORING, INTERNET OF THINGS (IoT), TELEGRAM BOT, WEB DASHBOARD, SENSOR NETWORK, MQ135, SHT31, LOGISTICS AUTOMATION, CARGO MONITORING.

Зміст

ВСТУП

У сучасній логістиці безпека та збереженість вантажів безпосередньо залежать від стабільності параметрів навколишнього середовища всередині транспортного засобу. Для багатьох категорій товарів, таких як продукти харчування, фармацевтичні препарати або високотехнологічна електроніка, навіть короткочасне відхилення від температурно-вологісної норми може призвести до повної втрати споживчих властивостей та значних фінансових збитків. Використання традиційних автономних логерів даних не завжди є ефективним, оскільки вони працюють у режимі накопичення інформації без можливості оперативного сповіщення персоналу в реальному часі. Тому розробка інтелектуальних систем моніторингу на базі енергоефективних мікроконтролерів із підтримкою сучасних бездротових технологій є необхідним кроком для цифровізації та автоматизації логістичних процесів.

Мета роботи полягає у створенні універсального апаратно-програмного комплексу на базі мікроконтролера ESP32, який здатний в автономному режимі контролювати мікроклімат вантажного відсіку, забезпечувати надійне зберігання історії показників на локальних носіях та надавати користувачу зручний інтерфейс для дистанційного керування та візуалізації даних у реальному часі.

Для успішної реалізації поставленої мети було визначено та виконано ряд послідовних науково-технічних задач. На початковому етапі проведено всебічний аналіз існуючих технічних рішень та методів дистанційного моніторингу фізичних параметрів середовища, що дозволило виявити актуальні вимоги до сучасних IoT-систем. На основі отриманих даних було науково обґрунтовано вибір елементної бази, зокрема використання високопродуктивного мікроконтролера ESP32 та набору прецизійних цифрових сенсорів, що забезпечують необхідну точність вимірювань.

Наступним кроком стала розробка комплексної архітектури системи, яка базується на інтеграції сенсорів через промислові інтерфейси I2C та OneWire, що стало фундаментом для проектування структурної та принципової електричної

схеми пристрою. В межах програмної реалізації було створено спеціалізоване забезпечення для стабільного збору даних з датчиків SHT31, DS18B20, MQ135 та BH1750. Особливу увагу приділено питанню відмовостійкості системи, що було досягнуто шляхом впровадження алгоритмів локального логування показників на microSD-карту у форматі CSV.

Окрім апаратної частини, було розроблено багаторівневий веб-інтерфейс (Dashboard) з вбудованою системою авторизації та проведено повну інтеграцію з месенджером Telegram для забезпечення миттєвого сповіщення користувачів про критичні зміни параметрів. На завершальному етапі роботи проведено серію експериментальних досліджень, результати яких дозволили оцінити ефективність функціонування розробленого прототипу в реальних умовах експлуатації.

Об’єктом дослідження є технологічні процеси дистанційного моніторингу фізичних параметрів середовища у закритих об’єктах та транспортних контейнерах.

Предметом дослідження виступають програмні та апаратні засоби побудови мобільної IoT-платформи на базі ESP32 для автоматизації контролю умов транспортування вантажів.

Практичне значення роботи полягає у створенні завершеного прототипу пристрою з гнучким налаштуванням через Wi-Fi (технологія WiFiManager), який може бути впроваджений приватними перевізниками та логістичними компаніями для мінімізації ризиків псування продукції та підвищення якості надання послуг.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ НАПРЯМУ РОЗРОБКИ

1.1. Аналіз проблем моніторингу умов транспортування вантажів

Сучасний етап розвитку глобальної логістики характеризується підвищенням вимог до якості та безпеки перевезень, особливо коли мова йде про специфічні категорії товарів. Своєчасна доставка вантажу більше не є єдиним критерієм ефективності, оскільки критичне значення має збереження його первинних фізико-хімічних властивостей протягом усього шляху від відправника до отримувача. Найбільш вразливими до зовнішніх впливів є фармацевтичні препарати, продукти харчування, що швидко псуються, а також прецизійне електронне обладнання. Для таких об'єктів навіть незначне відхилення від установленого температурного режиму або зміна рівня вологості може призвести до незворотних процесів розкладання, хімічних реакцій або механічних пошкоджень, що в кінцевому результаті спричиняє значні фінансові збитки та несе потенційну загрозу здоров'ю споживачів.

Однією з головних проблем існуючих систем контролю є їхня інертність та обмеженість у часі. Більшість доступних на ринку бюджетних рішень працюють за принципом автономного накопичення даних, які стають доступними для аналізу лише після прибуття транспортного засобу в пункт призначення та фізичного вилучення пристрою з вантажного відсіку. Такий підхід робить неможливим оперативне втручання в ситуацію, коли несправність холодильного обладнання або розгерметизація контейнера виникає на початку маршруту. Відсутність зворотного зв’язку в реальному часі створює інформаційний вакуум, у якому логісти та власники вантажу залишаються необізнаними про поточний стан товару до моменту його отримання, коли зазвичай фіксується вже доконаний факт псування.

Крім кліматичних факторів, суттєвою проблемою залишається контроль механічних впливів та безпеки доступу до вантажного простору. Випадкові удари під час завантаження, надмірна вібрація на неякісному дорожньому покритті або

несанкціоноване відкриття дверей можуть пошкодити крихкі елементи конструкцій або призвести до крадіжки майна. Традиційні засоби захисту, такі як механічні пломби, лише констатують факт порушення цілісності, але не дозволяють точно встановити час та місце події. Іншою важливою загрозою є накопичення шкідливих газів або задимлення всередині відсіку, що може бути ознакою як несправності транспортного засобу, так і початку пожежі всередині пакування.

Складність впровадження високотехнологічних систем моніторингу в автопарках часто зумовлена високою вартістю промислового обладнання та необхідністю створення спеціалізованої IT-інфраструктури для обробки сигналів. Більшість існуючих інтегрованих систем мають закриту архітектуру, що ускладнює їх масштабування або адаптацію під конкретні потреби малого та середнього бізнесу. Через це виникає об’єктивна потреба у створенні гнучкого, відносно недорогого та інтелектуального пристрою на базі сучасних мікроконтролерів. Така система має поєднувати в собі функції прецизійного вимірювання широкого спектра показників, локального резервування даних на незалежні носії та використання універсальних хмарних технологій для негайної передачі тривожних сповіщень безпосередньо відповідальним особам.

1.2. Огляд та порівняльний аналіз технологій моніторингу

Вибір технологічного стеку для побудови систем моніторингу вантажів залежить від балансу між вартістю розгортання, енергоефективністю та надійністю каналів передачі даних. Традиційні методи контролю, засновані на використанні пасивних RFID-міток, дозволяють ідентифікувати товар та фіксувати факт його проходження через контрольні точки, проте вони не здатні забезпечити безперервне спостереження за параметрами середовища всередині закритого контейнера під час руху. На відміну від них, активні датчики з підтримкою бездротових протоколів передачі дозволяють реалізувати повноцінну концепцію

інтернету речей, де кожен окремий пристрій стає повноправним вузлом глобальної мережі.

Серед технологій дальнього радіуса дії найбільш поширеними є рішення на базі стільникового зв'язку стандартів GSM та LTE, які забезпечують передачу даних практично з будь-якої точки світу. Однак впровадження таких модулів безпосередньо в мобільні системи моніторингу супроводжується низкою технічних та економічних складнощів, серед яких високе енергоспоживання пристрою в моменти активного пошуку сигналу, необхідність регулярної оплати послуг операторів зв'язку та складність підтримки стабільного з'єднання в умовах екранування вантажного відсіку металевим кузовом автомобіля. Альтернативні енергоефективні мережі великого радіуса дії, такі як LoRaWAN, демонструють чудові показники проникаючої здатності та автономності, проте вони потребують наявності розгалуженої мережі базових станцій (шлюзів) уздовж усього маршруту слідування, що значно обмежує їх застосування для міжнародних та міжрегіональних перевезень.

Технологія Wi-Fi, яка використана в даному проекті на базі мікроконтролера ESP32, виступає як найбільш універсальне та економічно виправдане рішення для систем локального та дистанційного моніторингу. Сучасні транспортні засоби все частіше оснащуються вбудованими точками доступу, а можливість використання смартфону водія в режимі модема дозволяє передавати дані з ESP32 на сервер без придбання додаткового дорогого обладнання. Гнучкість цього підходу полягає у можливості створення локальної веб-панелі безпосередньо на самому пристрої, що дозволяє отримувати доступ до показників через звичайний браузер у разі відсутності інтернету, а при наявності зовнішнього з’єднання — автоматично інтегруватися з хмарними сервісами та месенджерами.

Порівняно з промисловими стандартами передачі даних, використання відкритих API месенджерів, зокрема Telegram, є інноваційним підходом, що значно спрощує взаємодію людини з технічною системою. Це позбавляє необхідності розробляти спеціалізовані мобільні застосунки та підтримувати їх сумісність з різними операційними системами, оскільки вся логіка керування та сповіщення

реалізується на рівні бота. Таким чином, комбінація потужностей мікроконтролера ESP32 з протоколами Wi-Fi та хмарними інтерфейсами дозволяє створити високоефективну систему моніторингу, яка за функціональністю наближається до професійних логістичних рішень, при цьому залишаючись значно доступнішою за вартістю та простішою у повсякденній експлуатації.

1.3. Обґрунтування вибору елементної бази

Ефективність системи моніторингу вантажів залежить від правильного підбору апаратних компонентів, які мають забезпечувати високу точність вимірювань, надійність передачі даних та низьку вартість розробки. Центральним вузлом системи обрано мікроконтролер ESP32, який значно перевершує аналоги завдяки наявності двох обчислювальних ядер та інтегрованих модулів бездротового зв’язку Wi-Fi та Bluetooth. Його обчислювальна потужність дозволяє одночасно виконувати декілька критичних завдань: опитування широкого спектра сенсорів, підтримку роботи локального веб-сервера для візуалізації даних та забезпечення стабільного з’єднання з серверами Telegram для надсилання миттєвих сповіщень. Важливою перевагою є наявність енергонезалежної пам’яті Preferences, яка використовується в проекті для надійного зберігання ідентифікаторів користувачів Telegram та налаштувань оповіщень навіть після повного вимкнення живлення.

Для контролю мікроклімату у вантажному відсіку було обрано прецизійний цифровий сенсор SHT31, який працює за інтерфейсом I2C. Порівняно з поширеними датчиками серії DHT, він має значно меншу похибку при вимірюванні відносної вологості та температури, а також демонструє високу стабільність роботи в умовах агресивного середовища. Як додатковий інструмент термічного контролю в системі задіяно герметичний датчик DS18B20, що працює за протоколом OneWire. Його використання обумовлене можливістю винесення сенсора на значну відстань від основної плати, що дозволяє вимірювати температуру безпосередньо

всередині пакування вантажу, тоді як основний блок системи залишається в зоні стабільного радіозв’язку.

Моніторинг безпеки та цілісності вантажу реалізовано за допомогою датчика освітленості BH1750 та сенсора вібрації SW-420. Цифровий люксметр BH1750 забезпечує високу роздільну здатність при вимірюванні рівня світла, що дозволяє системі миттєво фіксувати несанкціоноване відкриття дверей вантажного відсіку або розгерметизацію контейнера. Датчик SW-420 у поєднанні з програмними алгоритмами обробки сигналів (дебаунсом) дає змогу вести точний облік механічних впливів та ударів, фіксуючи їхню кількість та інтенсивність протягом усього маршруту. Для виявлення хімічних забруднень повітря або задимлення інтегровано газоаналізатор MQ135, який через аналоговий вхід мікроконтролера дозволяє оцінювати загальну якість повітряного середовища.

Кінцевим елементом апаратної конфігурації є модуль microSD-карти та OLED-дисплей. Використання зовнішнього накопичувача пам’яті є критично важливим для забезпечення автономності системи, оскільки воно дозволяє вести безперервний запис логів у форматі CSV незалежно від наявності мережі Wi-Fi. У свою чергу, малогабаритний OLED-дисплей за технологією SSD1306 слугує основним засобом локальної діагностики, відображаючи поточну IP-адресу пристрою, стан підключення до мережі та критичні помилки модулів, що значно спрощує процес експлуатації системи водієм або технічним персоналом безпосередньо на місці використання.

1.4. Постановка задачі на розробку

На основі проведеного аналізу предметної області та обраної елементної бази, основною метою проекту є розробка автономного інтелектуального пристрою для безперервного контролю умов транспортування вантажів. Система повинна забезпечувати високоточне вимірювання температури, вологості, рівня освітленості та концентрації газів у режимі реального часу, а також фіксувати

механічні впливи на вантаж за допомогою датчика вібрації. Основним технічним завданням є створення програмних алгоритмів для стабільної роботи мікроконтролера ESP32 в багатозадачному режимі, що включає паралельне опитування сенсорів, відображення даних на локальному дисплеї та підтримку мережевих сервісів.

Важливою вимогою до розробки є забезпечення високого рівня відмовостійкості та автономності збереження даних. Для цього необхідно реалізувати функцію циклічного логування всіх отриманих показників на microSD-карту, що дозволить зберегти історію моніторингу навіть у зонах відсутності покриття Wi-Fi. Програмне забезпечення також має включати механізми автоматичної діагностики модулів та візуального оповіщення користувача про несправність будь-якого елемента системи.

Дистанційна частина задачі передбачає створення дворівневого інтерфейсу взаємодії з користувачем. Перший рівень – це захищена паролем веб-панель для детального моніторингу через браузер, яка також дозволяє динамічно змінювати налаштування Wi-Fi без перепрошивки пристрою. Другий рівень – інтегрований Telegram-бот, що виконує функції мобільного термінала для отримання звітів за запитом та миттєвого розсилання критичних сповіщень про порушення умов перевезення або несанкціонований доступ до вантажного відсіку. Таким чином, кінцевий продукт має бути завершеним програмно-апаратним рішенням, готовим до експлуатації в реальних логістичних умовах.

1.5. Висновки до першого розділу

У результаті проведеного аналізу встановлено, що забезпечення схоронності специфічних вантажів вимагає впровадження автоматизованих систем моніторингу, здатних працювати в режимі реального часу. Було визначено, що ключовими проблемами існуючих рішень є їхня висока вартість або відсутність

можливості миттєвого дистанційного сповіщення про критичні події. На основі порівняльного аналізу технологій бездротового зв’язку обґрунтовано використання мікроконтролера ESP32 та протоколу Wi-Fi як найбільш оптимальних за критеріями функціональності та економічної ефективності.

Визначено склад елементної бази, що включає високоточні цифрові сенсори SHT31 та DS18B20 для термічного контролю, а також датчики MQ135, BH1750 та SW-420 для комплексного моніторингу безпеки вантажного відсіку. Сформовано технічні вимоги до програмного забезпечення, які включають обов’язкове локальне резервування даних на microSD-карту та реалізацію дворівневого інтерфейсу керування через Web-панель та Telegram-бот. Це створює необхідний теоретичний та технічний фундамент для подальшої практичної реалізації системи.

РОЗДІЛ 2. РОЗРОБКА АПАРАТНО-ПРОГРАМНОЇ ЧАСТИНИ СИСТЕМИ

2.1. Побудова структурної схеми системи

Проектування апаратного забезпечення системи моніторингу вантажів базується на створенні ієрархічної структури, де центральним елементом є високоефективний мікроконтролер ESP32. Структурна схема, що відображає логічний взаємозв'язок між обчислювальним ядром та периферійними модулями, наведена на рисунку 2.1.

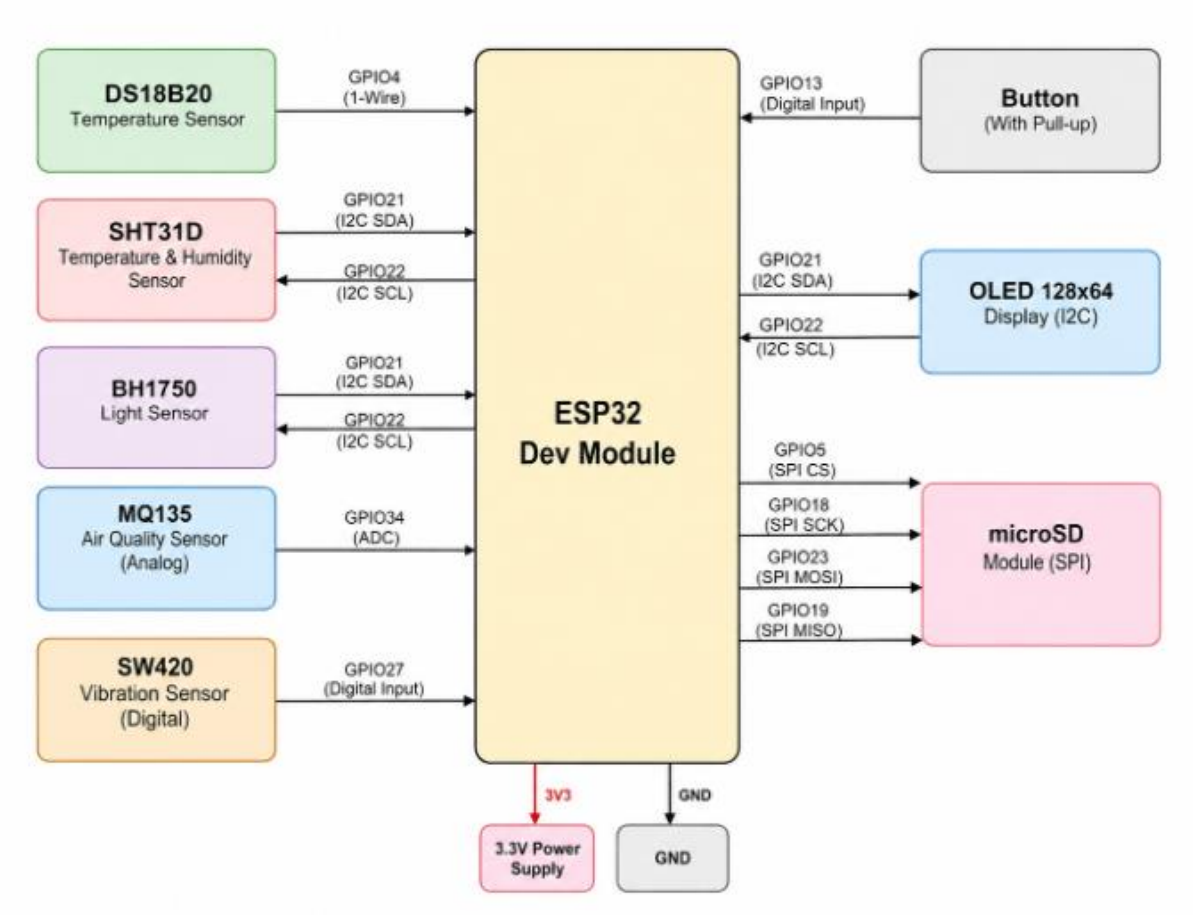

Рисунок 2.1 — Структурна схема апаратної частини системи моніторингу

Вона ілюструє підключення модулів, які відповідають за збір фізичних показників, локальне відображення інформації та її енергонезалежне зберігання. Основна концепція побудови полягає у використанні магістральних шин даних

(I2C та OneWire), що дозволяє паралельно підключити декілька цифрових пристроїв до обмеженої кількості виводів контролера, зберігаючи при цьому високу швидкість обміну інформацією та завадостійкість. Такий підхід забезпечує масштабованість системи, дозволяючи за необхідності додавати нові вимірювальні модулі без кардинальної зміни загальної архітектури комплексу.

2.2. Технічні характеристики та опис компонентів

Вибір елементної бази обумовлений жорсткими вимогами до експлуатації пристрою в умовах вібрацій та температурних перепадів, характерних для автомобільного транспорту. Кожен компонент системи пройшов перевірку на сумісність із логічними рівнями 3.3 В, що є стандартною напругою для сучасних енергоефективних мікроконтролерів.

2.2.1. Мікроконтролер ESP32-WROOM-32

Центральним обчислювальним вузлом системи обрано модуль ESP32-WROOM-32, зовнішній вигляд якого наведено на рисунку 2.2. Даний модуль інтегрує в собі двоядерний 32-бітний мікропроцесор Xtensa® Dual-core LX6. Вибір даного контролера обумовлений його високою продуктивністю та наявністю вбудованих апаратних засобів для реалізації концепції інтернету речей (IoT).

Рисунок 2.2 — Зовнішній вигляд мікроконтролера ESP32-WROOM-32

Архітектурна особливість використання двох ядер дозволяє ефективно розділити програмну логіку на рівні операційної системи реального часу (FreeRTOS). Одне ядро виділяється для забезпечення стабільної підтримки Wi-Fi з'єднання, обробки HTTP-запитів веб-сервера та взаємодії з API Telegram, тоді як інше ядро виконує детерміноване опитування датчиків у реальному часі, обробку зовнішніх переривань від датчика вібрації та запис даних на MicroSD-карту. Це виключає затримки в отриманні показників під час виконання мережевих операцій.

Наявність великої кількості GPIO-виводів та підтримка апаратних інтерфейсів SPI, I2C та UART дозволяє реалізувати паралельне підключення периферії без використання додаткових розширювачів портів. Вбудований 12-бітний аналогово-цифровий перетворювач (АЦП) використовується для обробки сигналу з газоаналізатора MQ135, що забезпечує достатню точність для моніторингу складу повітря. Основні технічні показники та можливості обраного модуля представлені у таблиці 2.1.

Таблиця 2.1 — Технічні характеристики контролера ESP32.

Мікроконтролер	ESP32 (2 ядра)
Частота	до 240 МГц
Пам’ять	520 КБ SRAM, 4 МБ Flash
Wi-Fi	802.11 b/g/n (2.4 GHz)
Bluetooth	4.2 (BLE)
GPIO	до 30
Інтерфейси	UART, SPI, I2C
Живлення	5V / 3.3V
Споживання	~80–260 мА
Режим сну	~10 мкА

Для практичної реалізації схеми підключення та повного розуміння апаратної топології пристрою необхідно детально розглянути призначення кожного фізичного виводу модуля. На рисунку 2.3 представлена технічна діаграма розпіновки (pinout), що відображає розширене мультиплексування функцій портів введення-виведення загального призначення (GPIO).

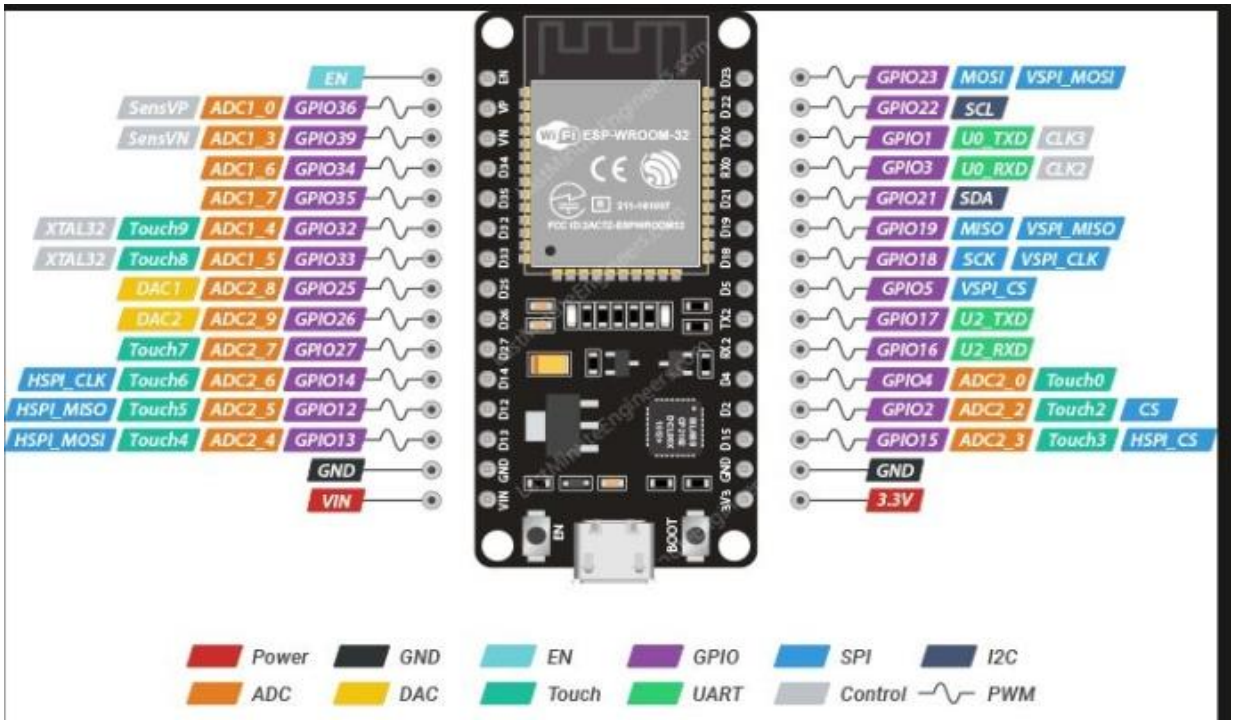

Рисунок 2.3 — Діаграма розпіновки (pinout) мікроконтролера ESP32-WROOM-32

Детальний аналіз апаратної конфігурації, що наведена на схемі розпіновки, дозволив оптимізувати використання обмежених ресурсів контролера для стабільної роботи всіх підсистем моніторингу. Для організації комунікації з прецизійними сенсорами SHT31 та BH1750 задіяно апаратну шину I2C через виводи GPIO 21, який виконує роль лінії даних SDA, та GPIO 22, що слугує лінією тактування SCL. Дані порти підтримують стабільний обмін на частоті до 400 кГц, що забезпечує високу швидкість зчитування показників без затримок у виконанні коду.

Отримання даних з аналогового газоаналізатора MQ135 реалізовано через пін GPIO 34. Вибір цього виводу є критично важливим, оскільки він належить до першого блоку АЦП (ADC1). Це дозволяє уникнути апаратних конфліктів, які виникають при спробі використання другого блоку АЦП одночасно з активним Wi-Fi з’єднанням, необхідним для роботи з API Telegram.

Процес локального логування даних на MicroSD-карту організовано через швидкісний інтерфейс SPI. У даній конфігурації використано виводи GPIO 18 для

тактування SCK, GPIO 19 для отримання даних MISO, GPIO 23 для передачі даних MOSI та GPIO 5 як лінію вибору пристрою CS. Такий розподіл дозволяє здійснювати потоковий запис інформації на зовнішній носій паралельно з іншими обчислювальними процесами.

Для фіксації механічних впливів датчик вібрації підключено до GPIO 4, який налаштовано на роботу в режимі зовнішніх переривань. Це дає змогу системі миттєво реагувати на тривожні події без необхідності постійного опитування стану входу в основному циклі програми, що значно підвищує загальну енергоефективність та швидкість відгуку пристрою. Використання виводів 3V3 та GND для живлення сенсорної периферії гарантує стабільність логічних рівнів та запобігає виходу з ладу чутливих напівпровідникових компонентів датчиків.

2.2.2. Цифрові сенсори температури та вологості SHT31 та DS18B20

Для забезпечення високої точності та надійності вимірювань у системі використано комбінацію двох типів термометрів, зовнішній вигляд яких представлено на рисунку 2.4. Це дозволяє контролювати як загальний стан середовища у відсіку, так і температуру безпосередньо всередині вантажу.

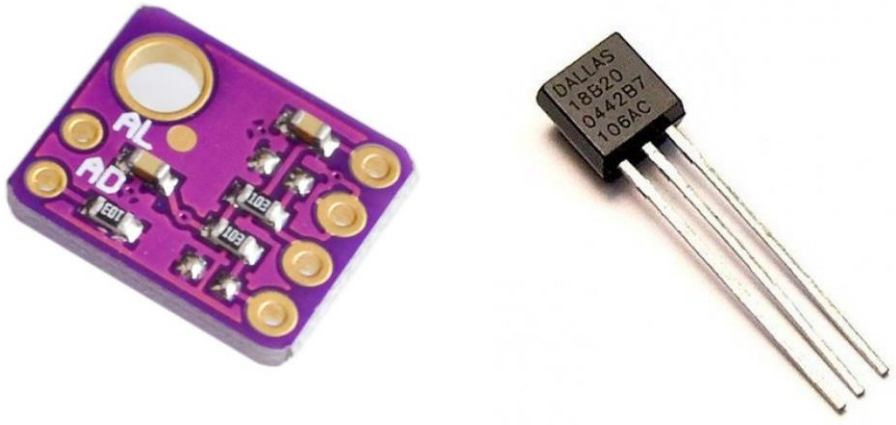

Рисунок 2.4 — Модулі датчиків SHT31 та DS18B20.

Сенсор SHT31 виступає основним вузлом контролю мікроклімату. Він демонструє мінімальну похибку та швидкий час відгуку завдяки інтегрованій схемі цифрової обробки сигналу на кристалі. Використання інтерфейсу I2C (Inter-Integrated Circuit) дозволяє підключати сенсор до загальної шини даних мікроконтролера, що спрощує апаратну реалізацію та забезпечує високу завадостійкість при передачі цифрового значення вологості та температури.

Додатково у системі задіяно датчик DS18B20 у герметичному виконанні. Його ключовою перевагою є робота за протоколом OneWire, який потребує лише однієї лінії зв’язку для обміну даними та живлення (режим паразитного живлення). Це дозволяє винести вимірювальний елемент у важкодоступні зони, підкладати його під пакування або розміщувати всередині контейнера на значній відстані від головного модуля, забезпечуючи точний моніторинг температури безпосередньо біля об’єкта контролю. Детальні порівняльні параметри обраних температурних сенсорів наведені у таблиці 2.2.

Таблиця 2.2 — Порівняльні параметри температурних сенсорів.

Параметр	**SHT31**	**DS18B20**
Вимірювання	Температура + вологість	Температура
Точність температури	±0.2°C	±0.5°C
Діапазон температур	-40…+125°C	-55…+125°C
Вологість	0–100% RH	✕ Немає
Інтерфейс	I2C	OneWire
Напруга живлення	2.4–5.5V	3.0–5.5V
Швидкість	Висока	Середня
Складність підключення	Середня	Проста

2.2.3. Напівпровідниковий газоаналізатор MQ135

Модуль MQ135 призначений для безперервного аналізу складу повітря та виявлення широкого спектру забруднюючих речовин у вантажному відсіку. Зовнішній вигляд датчика наведено на рисунку 2.5. Робота сенсора базується на зміні провідності діоксиду олова (SnO_2) при контакті з молекулами газів-відновників у чистому повітрі. Даний компонент ефективний для виявлення задимлення, а також концентрації аміаку (NH_3), бензолу, оксидів азоту (NO_x) та інших летких органічних сполук, що можуть свідчити про несправність транспортного обладнання або псування специфічних видів вантажу.

Рисунок 2.5 — Модуль датчика MQ135

Конструктивною особливістю сенсора є наявність вбудованого нагрівального елемента, який забезпечує необхідну робочу температуру для хімічної реакції на поверхні чутливого шару. Це зумовлює високе енергоспоживання модуля (до 200 мА), що було враховано при проектуванні системи живлення пристрою. Слід зазначити, що для отримання достовірних результатів сенсор потребує попереднього прогріву, під час якого вихідна напруга стабілізується.

Датчик передає аналоговий сигнал на вхід АЦП мікроконтролера, де отримане значення напруги перераховується у відносні одиниці концентрації газів (ppm) за допомогою логарифмічних залежностей, закладених у програмному забезпеченні. Передбачена можливість програмного калібрування сенсора в умовах чистого повітря для встановлення базового рівня (R0), що дозволяє компенсувати вплив температури та вологості навколишнього середовища на точність вимірювань. Основні технічні характеристики модуля MQ135 зведені у таблиці 2.3.

Таблиця 2.3 — Основні характеристики MQ135

Тип датчика	Газовий (якість повітря)
Виявляє гази	NH_3, NOx, CO_2, бензол, дим
Напруга живлення	5V
Вихід	Аналоговий (AO), цифровий (DO)
Діапазон вимірювання	~10–1000 ppm
Чутливість	Регулюється потенціометром
Час прогріву	20–60 с (рекомендовано більше)
Споживання	~150–200 мА
Робоча температура	-10…+50°C

2.2.4. Цифровий сенсор освітленості BH1750

Для реалізації функції контролю цілісності вантажного відсіку та виявлення фактів стороннього втручання до системи інтегровано цифровий люксметр BH1750, зовнішній вигляд якого представлено на рисунку 2.6. Використання даного сенсора дозволяє фіксувати факт несанкціонованого доступу до вантажу або розгерметизації контейнера через зміну рівня фонового освітлення.

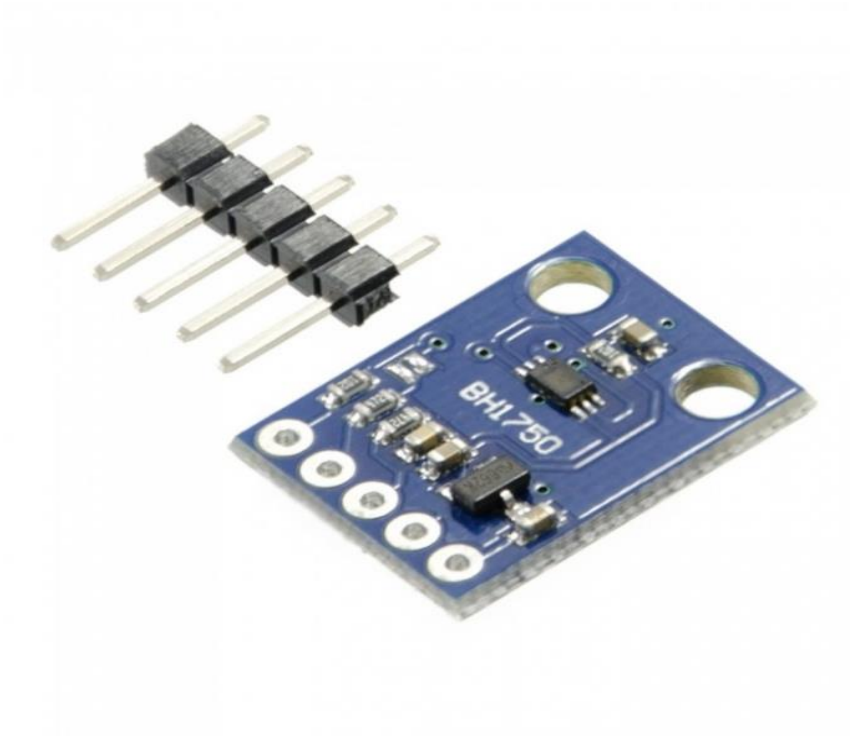

Рисунок 2.6 — Цифровий датчик освітленості BH1750

Сенсор побудований на основі фотодіода з інтегрованим підсилювачем та 16-бітним аналогово-цифровим перетворювачем, що забезпечує пряме виведення результату у люксах (lx) без необхідності складних обчислень на боці мікроконтролера. Ключовою особливістю BH1750 є його спектральна характеристика, яка максимально наближена до чутливості людського ока, що дозволяє коректно оцінювати рівень світла незалежно від типу джерела (природне або штучне освітлення).

Обмін даними здійснюється через послідовну шину I2C, що дозволяє підключати сенсор до тих самих ліній зв’язку, що і датчик SHT31, оптимізуючи використання GPIO-виводів ESP32. Програмна логіка системи налаштована на виявлення різких дельт освітленості: у разі перевищення встановленого порогу система миттєво генерує тривожне сповіщення та надсилає його користувачу через Telegram-бот. Такий підхід забезпечує високу швидкість реакції на інциденти безпеки під час транспортування. Основні технічні параметри та можливості сенсора BH1750 зведені у таблиці 2.4.

Таблиця 2.4 — Характеристики BH1750

Тип датчика	Освітленості (люксометр)
Діапазон вимірювання	1 – 65535 lx
Точність	±20%
Інтерфейс	I2C
Напруга живлення	3.3 – 5V
Роздільна здатність	до 1 lx
Час вимірювання	~120 мс
Споживання	~0.12 мА
Робоча температура	-40…+85°C

2.2.5. Модуль детекції вібрації та механічних впливів SW-420

Для забезпечення контролю якості транспортування та фіксації критичних механічних впливів у системі застосовується датчик вібрації SW-420, зовнішній вигляд якого наведено на рисунку 2.7. Його робота базується на використанні пружинного контакту, розміщеного всередині непровідної трубки. У стані спокою пружина замикає контакт, проте при виникненні вібрації, тряски або механічного удару пружина починає коливатися, що призводить до короткочасного розмикання електричного ланцюга.

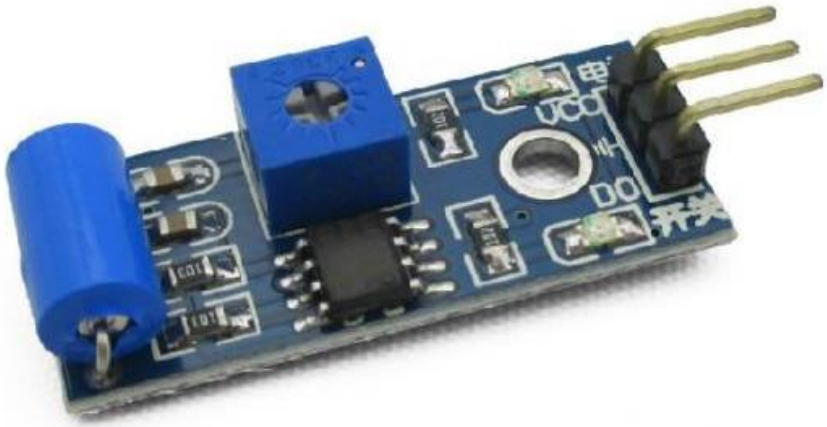

Рисунок 2.7 — Датчик вібрації SW-420.

Модуль оснащений компаратором LM393, який перетворює ці механічні коливання у стабільний цифровий сигнал (логічний рівень). Дискретний сигнал від датчика надходить на GPIO-вивід мікроконтролера, який налаштований на роботу в режимі зовнішнього переривання (External Interrupt). Це програмне рішення гарантує миттєву реєстрацію кожної події без затримок, оскільки обробка сигналу відбувається пріоритетно, незалежно від виконання основного циклу програми.

Важливою перевагою даного модуля є наявність вбудованого підстроювального резистора (потенціометра), що дозволяє регулювати поріг чутливості пристрою. Це дає можливість відфільтрувати природні вібрації автомобіля під час руху по нормальному дорожньому покриттю та реагувати виключно на сильні удари, падіння вантажу або нерівності дороги. Технічні характеристики обраного модуля датчика вібрації представлені у таблиці 2.5.

Таблиця 2.5 — Технічні характеристики модуля датчика вібрації SW-420

Тип датчика	Вібрації / удару
Принцип роботи	Механічний (пружинний контакт)
Напруга живлення	3.3 – 5V
Вихід	Цифровий (DO)
Чутливість	Регулюється потенціометром
Споживання	~5–15 мА
Час відгуку	< 10 мс
Робоча температура	-20…+70°C

2.2.6. Матричний OLED-дисплей на базі драйвера SSD1306

Для забезпечення оперативного контролю стану системи безпосередньо на місці її експлуатації використано графічний OLED-дисплей з діагоналлю 0.96 дюйма, зовнішній вигляд якого зображено на рисунку 2.8. На відміну від традиційних рідкокристалічних індикаторів, технологія органічних світлодіодів базується на використанні пікселів, що самовипромінюють світло, що усуває потребу в окремому модулі підсвітки. Це значно знижує загальне енергоспоживання пристрою та забезпечує високу чіткість і контрастність зображення, що є критичним для зчитування даних в умовах низької освітленості вантажних відсіків автомобіля.

Рисунок 2.8 — Зовнішній вигляд OLED-дисплея 0.96"

Дисплей побудований на базі драйвера SSD1306, який підтримує апаратне прискорення для виведення тексту та простої графіки. Підключення до мікроконтролера ESP32 здійснюється за допомогою послідовного інтерфейсу I2C (виводи SDA та SCL), що дозволяє мінімізувати кількість задіяних провідників та спростити топологію друкованої плати. Використання бібліотек Adafruit SSD1306 або U8g2 забезпечує гнучке керування буфером кадру, дозволяючи реалізувати плавне оновлення даних без видимого мерехтіння.

Програмна реалізація передбачає циклічне оновлення інформації на екрані, де відображаються поточні значення температури, вологості, рівня освітленості та концентрації газів. Завдяки високій контрастності матриці, виведені показники залишаються чіткими навіть при значних кутах огляду, що важливо для швидкої перевірки стану системи в обмеженому просторі вантажного відсіку. Окреме поле відведено для візуалізації мережевого статусу: при завантаженні системи на екран виводиться отримана IP-адреса пристрою, що дозволяє адміністратору швидко отримати доступ до локальної веб-панелі моніторингу. Окрім основних параметрів, на дисплей можуть виводитися сервісні повідомлення про стан підключення до Wi-Fi та активність Telegram-бота, що значно спрощує процес налагодження та

діагностики системи в польових умовах. Основні технічні характеристики дисплея зведені у таблиці 2.6.

Таблиця 2.6 — Технічні характеристики дисплея SSD1306

Тип дисплея	OLED
Контролер	SSD1315
Діагональ	0.96"
Роздільна здатність	128×64 пікселів
Колір	Жовто-синій
Інтерфейс	I2C
Напруга живлення	3.3 – 5V
Споживання	~20–30 мА
Кількість пінів	4 (VCC, GND, SDA, SCL)
Робоча температура	-40…+85°C

2.2.7. Модуль зчитування MicroSD-карт для реєстрації даних

Для забезпечення відмовостійкості системи та збереження цілісності моніторингових даних у проекті використано модуль MicroSD-адаптера, зовнішній вигляд якого представлено на рисунку 2.9. Цей компонент виконує функцію локального сховища «чорної скриньки», що є критично важливим при транспортуванні вантажів у зонах із нестабільним покриттям Wi-Fi або під час руху транспортного засобу поза межами дії мережі.

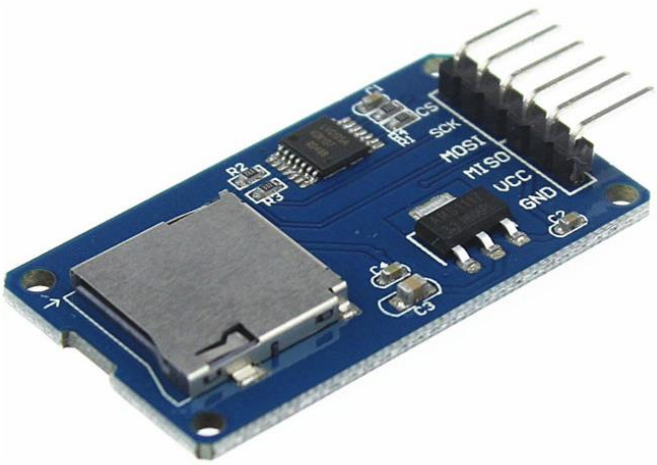

Рисунок 2.9 — Модуль адаптера MicroSD-карти.

Обмін даними між мікроконтролером ESP32 та картою пам'яті здійснюється за допомогою послідовного периферійного інтерфейсу SPI. Модуль оснащений вбудованим стабілізатором напруги та перетворювачем логічних рівнів, що забезпечує безпечне узгодження 5-вольтового живлення із 3.3-вольтовою логікою шини даних.

Безпосереднім носієм інформації виступає карта пам'яті формату MicroSDHC об'ємом 16 ГБ (Class 10), що забезпечує високу швидкість запису та достатній ресурс пам'яті для тривалого автономного логування. Програмна логіка реалізує запис інформації у текстовому форматі CSV (Comma-Separated Values) на файлову систему FAT32. Кожен запис містить часову мітку та актуальні показники всіх сенсорів (температури, вологості, газу, світла та вібрації), що дозволяє легко імпортувати дані у табличні редактори для подальшої аналітики та побудови графіків після завершення рейсу. Основні технічні параметри модуля реєстрації даних наведені у таблиці 2.7.

Таблиця 2.7 — Характеристики модуля реєстрації даних

Тип модуля	microSD (зберігання даних)
Інтерфейс	SPI
Напруга живлення	4.5 – 5.5 В

Логічні рівні	3.3В / 5В
Стабілізатор	AMS1117-3.3
Перетворювач рівнів	74LVC125
Споживання струму	0.2 – 200 мА
Підтримка карт	microSD (≤2 ГБ), microSDHC (≤32 ГБ)
Розміри	42×24×12 мм
Вага	~5 г

2.2.8. Комутаційні елементи та засоби монтажу

Для забезпечення надійного електричного з'єднання між усіма функціональними модулями системи використано набір спеціалізованих гнучких провідників типу «тато-мама» та «мама-мама» (DuPont wires), приклад яких наведено на рисунку 2.10. Ці провідники є стандартом для прототипування електронних пристроїв завдяки кроку роз'ємів 2.54 мм, що забезпечує повну сумісність із контактними групами мікроконтролера ESP32 та периферійних сенсорів.

Рисунок 2.10 — Набір з’єднувальних провідників для макетування

Конструктивно провідники складаються з багатожильного мідного осердя у ПВХ-ізоляції, що забезпечує необхідну стійкість до втоми металу при багаторазових перегинах та вібраційних навантаженнях, характерних для експлуатації всередині вантажного автомобіля. Щільна посадка роз'ємів запобігає випадковому розмиканню контактів при механічній трясці. Варто враховувати, що для забезпечення цілісності сигналів високошвидкісних інтерфейсів (I2C, SPI) довжина таких провідників не повинна перевищувати 20 см. Це обмеження дозволяє мінімізувати вплив паразитної ємності та наведень, які можуть призвести до помилок при передачі даних від сенсорів до центрального процесора.

Особливу увагу приділено колірному маркуванню ліній зв’язку, що мінімізує ризик виникнення короткого замикання при монтажі:

- Червоний та чорний кольори зарезервовані виключно для ліній живлення (VCC) та заземлення (GND) відповідно.
- Яскраві кольори (жовтий, синій, зелений) використані для диференціації сигнальних ліній цифрових інтерфейсів I2C та SPI.

Такий системний підхід до комутації не лише спрощує первинне збирання прототипу, але й значно підвищує ремонтопридатність системи, дозволяючи швидко ідентифікувати та замінити пошкоджену лінію в польових умовах.

2.2.9. Безпаячна макетна плата для прототипування

Для фіксації мікроконтролера та організації спільних шин живлення і заземлення в системі використано безпаячну макетну плату (breadboard), зовнішній вигляд якої представлено на рисунку 2.11. Вона має внутрішню структуру з металевих затискачів, що дозволяє швидко та надійно з’єднувати компоненти за допомогою дротів-перемичок. Використання такої плати є критично важливим на етапі проектування, оскільки це дає змогу оперативно змінювати схему підключення датчиків або додавати нові модулі без ризику пошкодження компонентів перегрівом при паянні.

Рисунок 2.11 — Зовнішній вигляд макетної плати (breadboard).

Конструкція плати передбачає наявність окремих рейок живлення, що проходять вздовж всього корпусу, що дозволило централізовано підключити лінію 3.3 В до всіх сенсорів системи. Це мінімізує кількість дротів, що йдуть безпосередньо від мікроконтролера, та робить конструкцію більш впорядкованою та зручною для діагностики під час тестування у вантажному відсіку.

2.3. Розробка принципової електричної схеми підключення

Розробка принципової електричної схеми є визначальним етапом проектування, оскільки вона встановлює фізичні та логічні зв’язки між обчислювальним ядром системи та периферійним обладнанням. Повна принципова електрична схема розробленої системи моніторингу представлена на рисунку 2.12. У даному проекті за основу обрано мікроконтролер ESP32, що зумовлено його високою продуктивністю, наявністю інтегрованих бездротових інтерфейсів та великою кількістю універсальних портів введення-виведення (GPIO). Процес проектування схеми вимагав ретельного розподілу ресурсів контролера для забезпечення стабільної роботи трьох незалежних цифрових інтерфейсів та аналогового каналу вимірювання, уникаючи при цьому апаратних конфліктів.

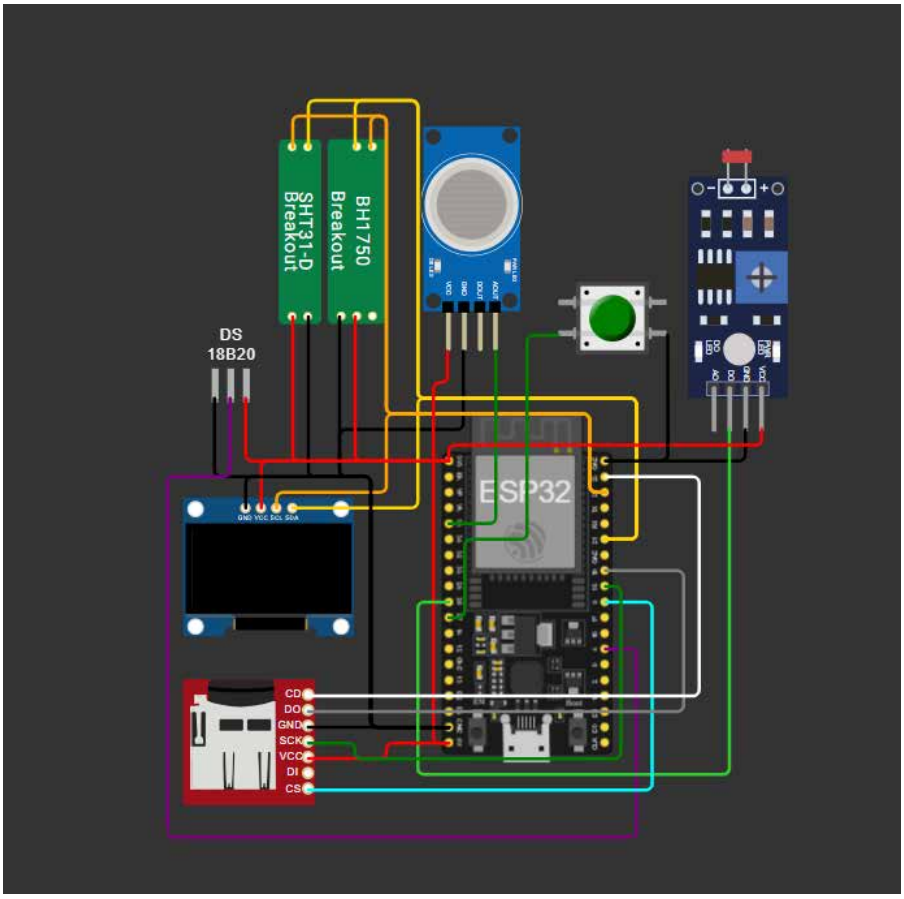

Рисунок 2.12 — Принципова електрична схема системи моніторингу вантажів.

Центральною частиною схеми є шина I2C (Inter-Integrated Circuit), яка функціонує за двопровідною топологією (лінії SDA на GPIO 21 та SCL на GPIO 22). Вибір цього інтерфейсу дозволив реалізувати концепцію спільного використання шини, де до однієї пари контактів паралельно підключено одразу три пристрої: інтелектуальний сенсор мікроклімату SHT31, датчик інтенсивності освітлення BH1750 та графічний OLED-дисплей для локального моніторингу даних. Кожен з цих модулів має унікальну апаратну адресу, що дозволяє мікроконтролеру здійснювати вибіркове опитування сенсорів без перехресних завад. Таке рішення забезпечує високу масштабованість системи та значну економію вільних виводів ESP32.

Паралельно з низькошвидкісною шиною I2C, у схемі реалізовано високошвидкісний інтерфейс SPI (Serial Peripheral Interface), призначений для

взаємодії з модулем карти пам’яті MicroSD. Оскільки процес логування телеметрії передбачає регулярний запис масивів даних на зовнішній носій, використання протоколу SPI є технічно обґрунтованим через його повнодуплексну архітектуру та високу пропускну здатність. Підключення здійснено до апаратного стеку VSPI (GPIO 18, 19, 23), а керування вибором пристрою (Chip Select) закріплено за GPIO 5. Це гарантує, що операції запису на карту не будуть блокувати виконання інших програмних процесів, забезпечуючи цілісність файлової системи навіть при інтенсивному потоці даних.

Для розширення функціональних можливостей моніторингу до схеми інтегровано спеціалізовані сенсори з різними принципами передачі сигналу. Зокрема, цифровий термометр DS18B20 підключено за протоколом OneWire до GPIO 4. Особливістю цього вузла є використання зовнішнього підтягуючого резистора номіналом 4.7 кОм між лінією даних та шиною живлення 3.3 В, що є необхідною умовою для стабілізації сигналу та коректної дешифрації часових слотів протоколу при використанні довгих з’єднувальних дротів. Окрему увагу приділено безпековому контуру: датчик вібрації SW-420 підключено до GPIO 33 у режимі цифрового входу, що дозволяє системі миттєво реагувати на зовнішні механічні впливи. Аналоговий сегмент схеми представлений газоаналізатором MQ-135, вихід якого під’єднано до 12-бітного аналого-цифрового перетворювача (ADC1, GPIO 34). Це дозволяє перетворювати концентрацію домішок у повітрі в цифрові значення для подальшої математичної обробки та візуалізації.

Завершальним аспектом розробки схеми є організація єдиного контуру живлення. Вся периферія та мікроконтролер працюють від напруги 3.3 В, що повністю нівелює потребу у використанні перетворювачів логічних рівнів та спрощує архітектуру пристрою. Для фільтрації імпульсних завад, що виникають під час роботи WiFi-модема, схема передбачає встановлення блокувальних конденсаторів безпосередньо біля виводів живлення активних модулів. Таким чином, розроблена принципова схема забезпечує надійну електричну взаємодію всіх компонентів, мінімізує енерговитрати та створює стійку базу для програмної реалізації системи моніторингу.

2.4. Обґрунтування системи живлення пристрою

Для забезпечення стабільної та безперебійної роботи розробленої системи моніторингу проведено детальний аналіз енергетичного балансу всіх її компонентів. Основним викликом при проектуванні схеми живлення стала необхідність одночасного забезпечення енергією як малопотужних цифрових сенсорів, так і енергоємних вузлів, таких як нагрівальний елемент газового датчика та радіомодуль зв'язку.

Центральний обчислювальний модуль ESP32 функціонує в режимі постійної активності, оскільки він одночасно обробляє дані з п'яти різнотипних датчиків, підтримує роботу локального Web-сервера та Telegram-бота для дистанційного керування. Найбільше навантаження на систему живлення виникає під час використання Wi-Fi стека, коли пікові значення струму можуть сягати 240-260 мА. Поряд із цим, значну частку загального споживання складає датчик якості повітря MQ135, який для коректного вимірювання вмісту газів потребує постійного розігріву внутрішнього сенсора, що додає до базового навантаження близько 150 мА.

Процес логування даних на microSD-карту, який за програмою відбувається кожні 5 секунд, також супроводжується короткочасними сплесками енергоспоживання через активацію шини SPI та механізми запису файлової системи. На противагу їм, цифрові датчики температури та вологості SHT31, освітленості BH1750 та виносний термометр DS18B20 працюють за енергоефективними протоколами I2C та OneWire, сумарно споживаючи менше 5 мА, що робить їх вплив на загальний бюджет потужності мінімальним. Також до системи інтегровано датчик вібрації SW-420, який перебуває в режимі очікування переривань і майже не впливає на розряд джерела живлення.

Візуалізація стану системи на OLED-дисплеї SSD1306 потребує близько 20 мА, проте це значення може змінюватися залежно від кількості активних пікселів

під час прокручування тексту в меню. Сумарний розрахунковий струм системи у пікових режимах становить близько 450-500 мА. Враховуючи необхідність запасу потужності для стабільної роботи без просадок напруги, що могли б призвести до програмних помилок або втрати зв'язку, було обрано джерело живлення з номінальним струмом 1 А та напругою 5 В. Використання такої конфігурації дозволяє нівелювати вплив перехідних процесів під час активації microSD-модуля та забезпечує стабільну роботу Telegram-сповіщень навіть при низькій якості сигналу мережі. Для фільтрації високочастотних завад, що виникають під час роботи радіомодуля, у ланцюгах живлення передбачено використання розв’язуючих конденсаторів. Детальні показники енергоспоживання кожного вузла системи зведені у таблиці 2.2.

Таблиця 2.2 — Енергетичне споживання компонентів системи

Найменування компонента	Робоча напруга, В	Струм споживання (середній), мА	Струм споживання (піковий), мА
Контролер ESP32 (Wi-Fi активний)	3,3	80	260
Датчик якості повітря MQ-135	5,0	150	160
Модуль microSD (запис)	3,3	50	80
OLED-дисплей SSD1306	3,3	10	20

Найменування компонента	Робоча напруга, В	Струм споживання (середній), мА	Струм споживання (піковий), мА
Разом (пікове навантаження)	-	-	~520

2.5. Конструктивне виконання та компонування системи

Проектування конструкції системи моніторингу здійснювалося з урахуванням специфіки експлуатації у вантажних автомобілях, що передбачає постійні вібрації, обмежений простір та необхідність швидкого доступу до інтерфейсів керування. Основним конструктивним рішенням стало розділення системи на центральний блок керування та виносні вимірювальні модулі, що дозволяє досягти гнучкості при монтажі пристрою в кабіні або вантажному відсіку. Центральний вузол системи базується на мікроконтролері ESP32, який разом із модулем запису на microSD-карту та OLED-дисплеєм розміщений у захисному корпусі, що забезпечує жорстку фіксацію електронних компонентів та захист від механічних пошкоджень.

Особлива увага при компонуванні була приділена розміщенню сенсорної мережі для отримання максимально точних даних. Датчик вібрації та ударів SW-420 жорстко закріплений на внутрішній основі корпусу, що забезпечує пряму передачу механічних імпульсів від транспортного засобу до сенсора без демпфування. На противагу цьому, високоточний цифровий термометр DS18B20 реалізований у вигляді виносного герметичного щупа, підключеного через окремий роз'єм, що дає можливість контролювати температуру безпосередньо всередині упаковки вантажу або в найбільш критичних точках контейнера. Датчики навколишнього середовища, такі як SHT31 для вимірювання вологості та MQ135

для аналізу якості повітря, розташовані у верхній частині корпусу з передбаченими вентиляційними отворами для вільної циркуляції повітря навколо вимірювальних камер.

Ергономіка пристрою забезпечує зручність взаємодії для водія або логіста без використання додаткових гаджетів. На передню панель винесено OLED-дисплей, який у режимі реального часу відображає поточні показники сенсорів, мережевий статус та IP-адресу пристрою, а керування інтерфейсом здійснюється за допомогою однієї багатофункціональної кнопки з програмним захистом від випадкових натискань. Слот для карти пам'яті розташований на бічній грані пристрою для забезпечення оперативного вилучення носія з накопиченими логами після завершення рейсу. Внутрішній монтаж виконаний з використанням роз’ємних з’єднань, що підвищує ремонтопридатність системи та дозволяє проводити швидку заміну окремих вузлів у разі їх виходу з ладу в процесі експлуатації. Зовнішній вигляд розробленого діючого прототипу системи на етапі макетування та тестування функціоналу представлено на рисунку 2.13.

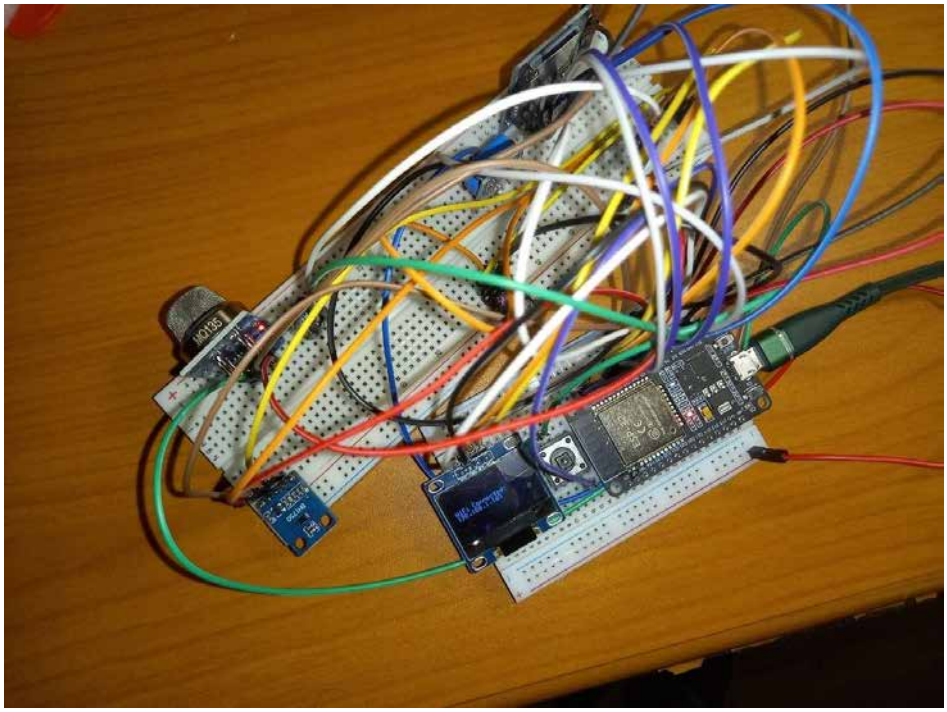

Рисунок 2.13 — Загальний вигляд працюючого макета системи

2.6. Програмне забезпечення системи моніторингу та алгоритми її функціонування

Програмний комплекс системи моніторингу розроблений на мові програмування C++ з використанням фреймворку Arduino для платформи ESP32. Вибір даного стеку технологій обумовлений необхідністю високої швидкості обробки сигналів у реальному часі та нативною підтримкою багатозадачності через операційну систему FreeRTOS, яка інтегрована в ядро ESP32. Це дозволяє розділяти критичні завдання (зчитування показників сенсорів) та фонові процеси (підтримка Wi-Fi з'єднання та обробка Telegram-запитів) без взаємного блокування.

Основна концепція архітектури базується на принципі модульності. Це означає, що кожен сенсор та мережевий інтерфейс обслуговуються окремими програмними об’єктами. Такий підхід полегшує діагностику, дозволяє проводити ізольоване тестування кожного вузла та забезпечує гнучкість при подальшому розширенні функціоналу системи. Програмний код інтегрує роботу з апаратними протоколами I2C, OneWire, SPI та стеком протоколів TCP/IP.

Для розробки та відладки було використано середовище Arduino IDE 2.x, інтерфейс якого під час написання коду для модуля MQ135 представлено на рисунку 2.14. Процес розробки включав етапи конфігурації менеджеру платформ для підтримки мікроконтролерів серії ESP32 та підключення спеціалізованих зовнішніх бібліотек для роботи з периферією. Взаємодія з месенджером Telegram реалізована через асинхронні запити, що забезпечує стабільну роботу системи навіть при тимчасових затримках у мережі.

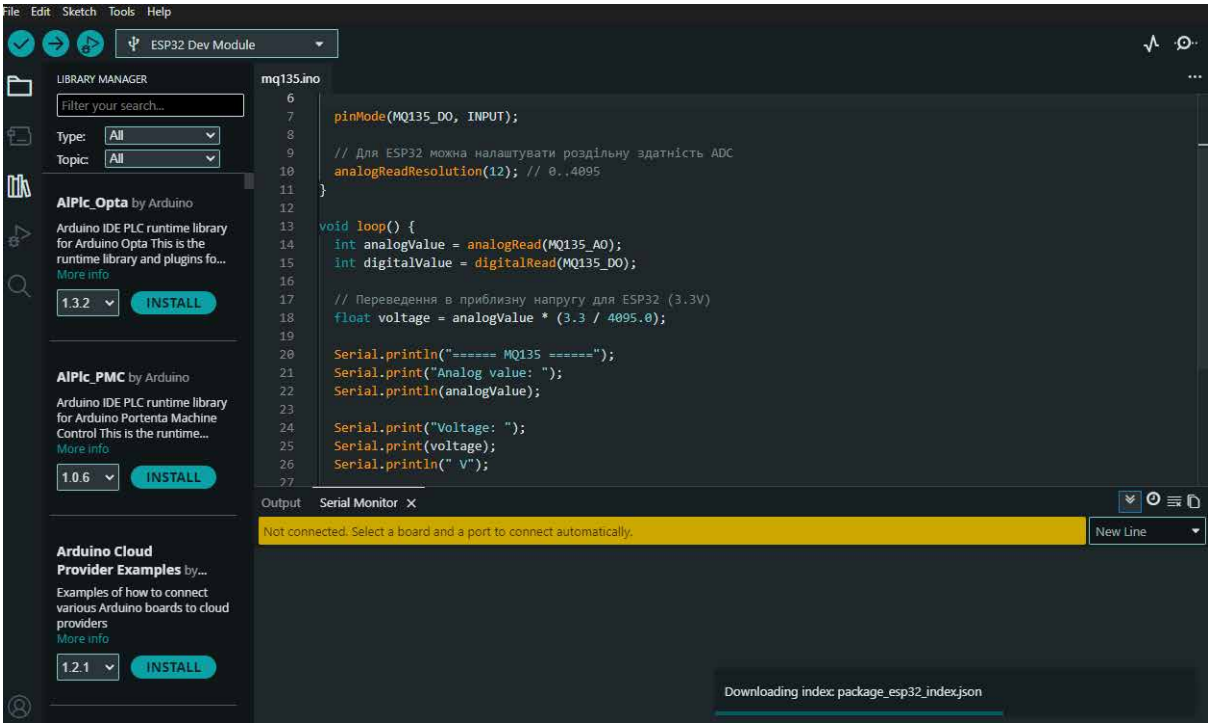

Рисунок 2.14 — Вікно монітора послідовного порту Arduino IDE з результатами ініціалізації системи

Перелік та призначення основних програмних бібліотек, що складають основу програмного забезпечення, наведено у таблиці 2.3.

Таблиця 2.3 — Програмні бібліотеки, використані при розробці системи

Бібліотека	**Призначення**	**Функціональна роль**
WiFi.h	Мережевий стеку	Забезпечення підключення до локальної мережі Wi-Fi.
WiFiManager.h	Мережева конфігурація	Створення веб-інтерфейсу для налаштування SSID/Password.

Бібліотека	**Призначення**	**Функціональна роль**
Adafruit_SHT31.h	Датчик SHT31	Отримання точних значень температури та вологості (I2C).
DallasTemperature.h	Датчик DS18B20	Робота з герметичним температурним зондом (OneWire).
Adafruit_BH1750.h	Датчик люксметра	Вимірювання рівня освітленості в контейнері.
UniversalTelegramBot.h	Telegram Bot API	Реалізація віддаленого керування та системи сповіщень.
ArduinoJson.h	Обробка даних	Парсинг JSON-відповідей від серверів та Telegram API.
Adafruit_SSD1306.h	OLED-дисплей	Виведення системної інформації на екран пристрою.

Бібліотека	Призначення	Функціональна роль
SD.h / SPI.h	Файлова система	Керування процесом логування даних на microSD-карту.

2.6.1 Алгоритми підключення до мережі та ініціалізація периферії

Процес функціонування системи моніторингу розпочинається з виконання процедури ініціалізації апаратної частини у функції setup(). На початковому етапі активується послідовна шина I2C із заданими параметрами пінів OLED_SDA (21) та OLED_SCL (22), що є необхідною умовою для запуску графічного OLED-дисплея на базі контролера SSD1306 за адресою 0x3C. Після очищення відеобуфера на екран виводиться сервісне повідомлення «System starting...», яке підтверджує успішний старт мікроконтролера. Далі програмний алгоритм почергово опитує цифрові інтерфейси сенсорів освітленості BH1750 та температури/вологості SHT31, причому для останнього реалізовано механізм автоматичного визначення I2C-адреси (0x44 або 0x45), що підвищує відмовостійкість системи.

Паралельно з опитуванням I2C-периферії виконується конфігурація цифрових пінів для роботи з датчиком вібрації SW-420 та кнопкою керування інтерфейсом у режимі INPUT_PULLUP. Згідно з логікою, наведеною на рисунку 2.16, наступним кроком є ініціалізація роботи з microSD-картою через інтерфейс SPI (пін SD_CS_PIN 5). У разі успішного монтування файлової системи алгоритм автоматично створює або відкриває файл log.csv та записує до нього заголовок із переліком параметрів, що підлягають логуванню, включаючи показники якості повітря, температури, вологості та інтенсивності вібрацій.

Ключовим етапом завантаження є виконання функції connectWiFi(), де інтегровано алгоритм адаптивного підключення на базі об'єкта WiFiManager.

Програмна логіка передбачає спробу автоматичного з'єднання з раніше збереженою в пам'яті точкою доступу. У випадку відсутності зв'язку мікроконтролер переходить у режим програмної точки доступу (Soft-AP) з ідентифікатором «ESP32-Cargo-Setup», відображаючи на OLED-дисплеї інструкцію щодо підключення та локальну адресу (192.168.4.1) для входу в інтерфейс налаштувань. Після встановлення з’єднання на екран виводиться отримана від роутера динамічна IP-адреса, що дозволяє користувачеві перейти до основної веб-панелі моніторингу, що зображено на рисуноку 2.15.

Рисунок 2.15 — Відображення поточної IP-адреси пристрою на OLED-дисплеї

Після встановлення з’єднання система ініціалізує веб-сервер на стандартному порту 80, завантажує персоналізовані налаштування Telegram-користувачів через бібліотеку Preferences та надсилає сервісне сповіщення про готовність до роботи. Повна логічна послідовність дій алгоритму мережевого підключення та ініціалізації представлена на блок-схемі на рисуноку 2.16.

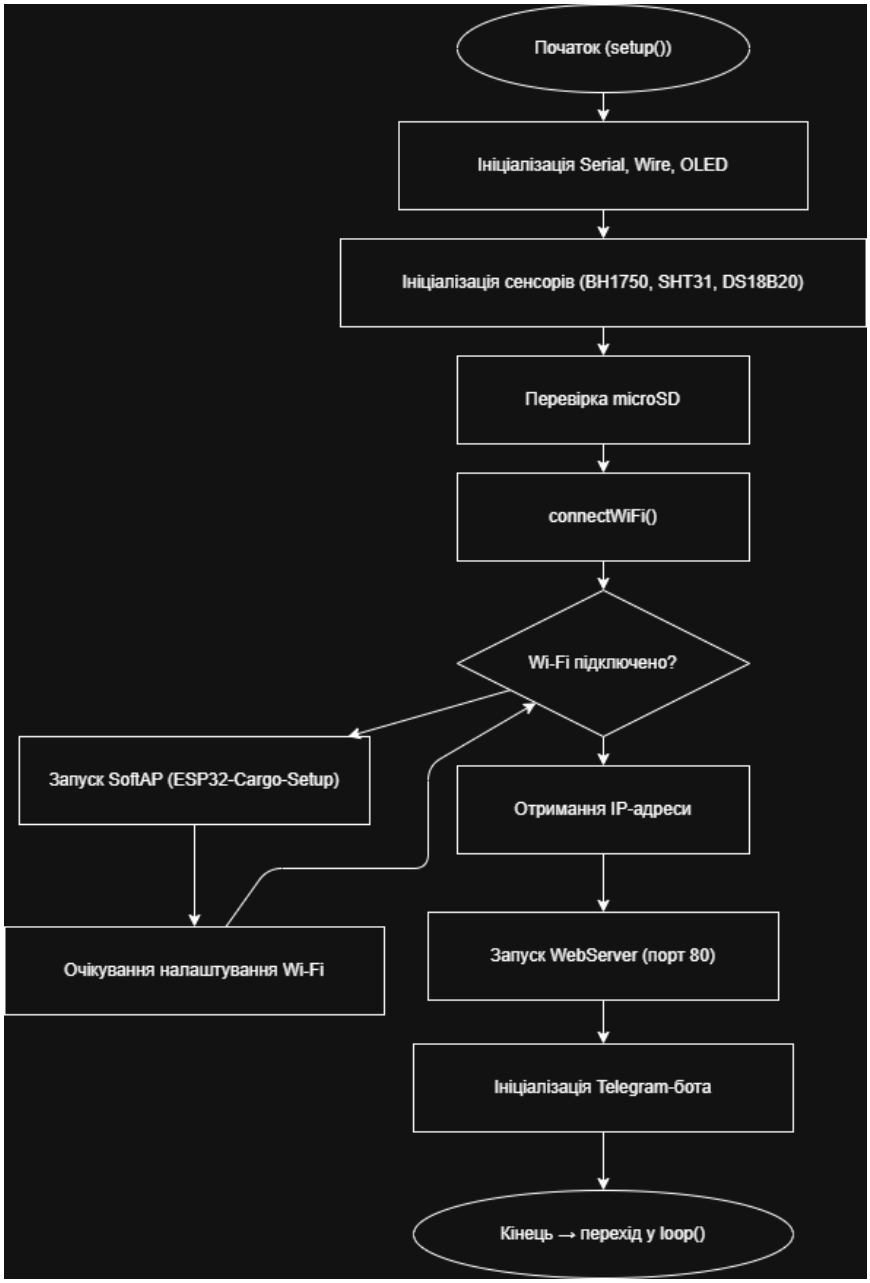

Рисунок 2.16 — Блок-схема алгоритму мережевого підключення та ініціалізації системи

2.6.2. Програмна реалізація збору, обробки та аналізу сенсорних даних

Логіка функціонування системи в межах головного циклу loop() базується на принципі періодичного опитування сенсорів із заданим інтервалом, що забезпечується функцією millis() для уникнення блокування процесора. Для

аналізатора газів MQ135 реалізовано алгоритм перетворення аналогового сигналу, отриманого з 12-бітного АЦП мікроконтролера, у напругу за формулою:

$$V_{out} = \frac{ADC_{val} \times 3.3}{4095}$$

після чого виконується обчислення відсоткового еквівалента забруднення щодо базового рівня чистого повітря. Програмний алгоритм порівнює отримані значення з критичним порогом (параметр gasThreshold), і у разі його перевищення система ініціює переривання нормального режиму роботи, змінюючи статус на дисплеї та активуючи процедуру негайного сповіщення через Telegram-бот. Конфігураційні параметри цього та інших критичних станів системи, а також відповідні дії контролера наведені у таблиці 2.5.

Таблиця 2.5 — Конфігураційні параметри критичних станів системи

Параметр	Порогове значення	Одиниця виміру	Дія системи
Концентрація газів	> 400 (ум. од.)	ADC units	Telegram Alert / OLED Warning
Рівень вібрації	> 10	Спрацювань/хв	Запис події в лог
Температура (max)	+45	°C	Попередження на дисплеї

Обробка даних від цифрових датчиків SHT31 та DS18B20 передбачає механізм валідації, де перед записом у лог-файл перевіряється діапазон отриманих значень на відповідність фізично можливим параметрам середовища. Це дозволяє мінімізувати вплив апаратних збоїв шини I2C або OneWire на загальну статистику моніторингу. Особливу увагу приділено обробці сигналів від датчика вібрації SW-420, для якого реалізовано програмний фільтр брязкоту контактів (debounce).

Алгоритм використовує інкрементальний лічильник механічних впливів, який накопичує кількість спрацювань за певний проміжок часу, що дає змогу оцінити не просто наявність вібрації, а її інтенсивність та потенційну небезпеку для цілісності вантажу.

Фінальний етап обробки даних включає агрегацію всіх показників у структурований рядок формату CSV для подальшого логування на карту пам’яті microSD та передачу даних на веб-панель за допомогою JSON-запитів. Для забезпечення взаємодії з користувачем реалізовано обробник команд Telegram-бота, який працює в асинхронному режимі. Це дозволяє системі одночасно виконувати моніторинг середовища та відповідати на запити користувача. Зокрема, за запитом команди /sensors система миттєво формує та надсилає звіт із актуальними даними всіх підключених модулів, що продемонстровано на рисунку 2.17.

Рисунок 2.17 — Інтерфейс взаємодії з користувачем через Telegram-бот

Логічна структура основного циклу збору даних візуалізована за допомогою блок-схеми на рисунку 2.17.

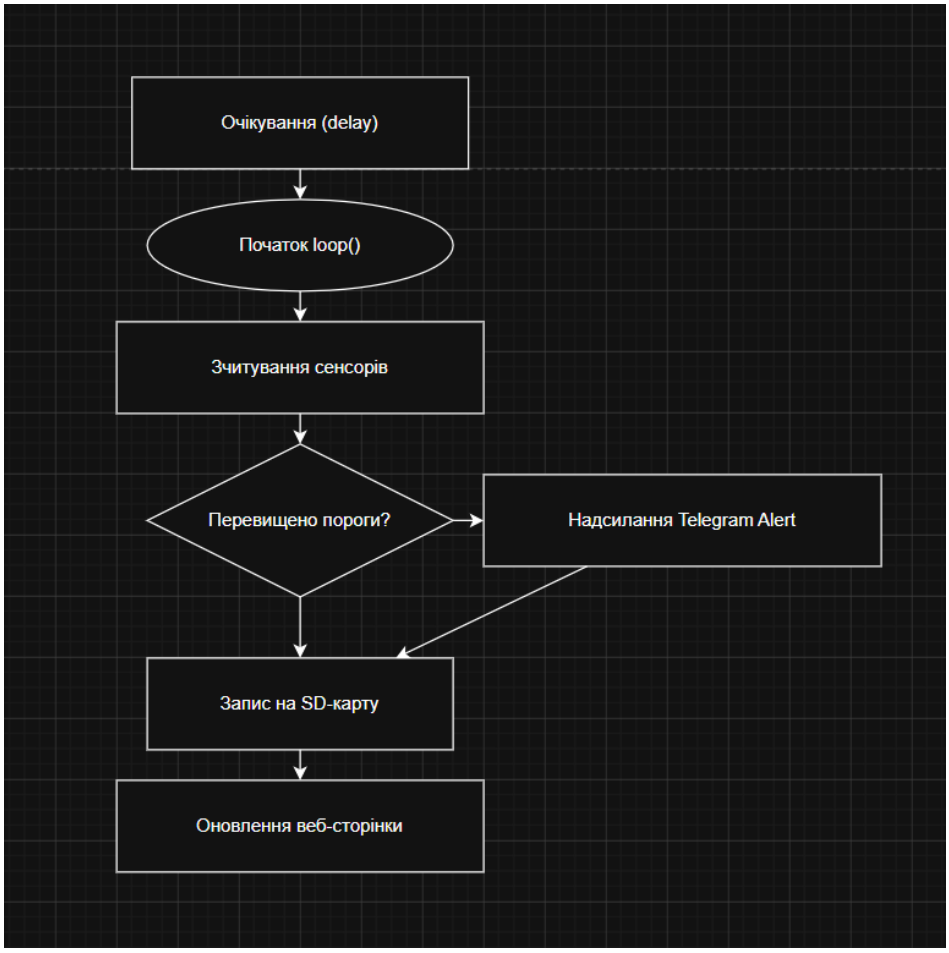

Рисунок 2.18 Блок-схема циклу Loop

2.6.3 Розробка вебсервера та інтеграція з месенджером Telegram

Програмне забезпечення системи містить вбудований асинхронний вебсервер, який забезпечує пряму взаємодію користувача з пристроєм через захищену паролем панель керування, що розгорнута безпосередньо на мікроконтролері ESP32. Веб-інтерфейс реалізований за допомогою технологій AJAX та JSON, що дозволяє динамічно оновлювати показники сенсорів на сторінці моніторингу в режимі реального часу без її повного перезавантаження. Такий підхід мінімізує обчислювальне навантаження на головний процесор та значно знижує обсяг мережевого трафіку в локальній мережі. Зовнішній вигляд розробленої вебпанелі при перегляді через браузер представлено на рисунку 2.19.

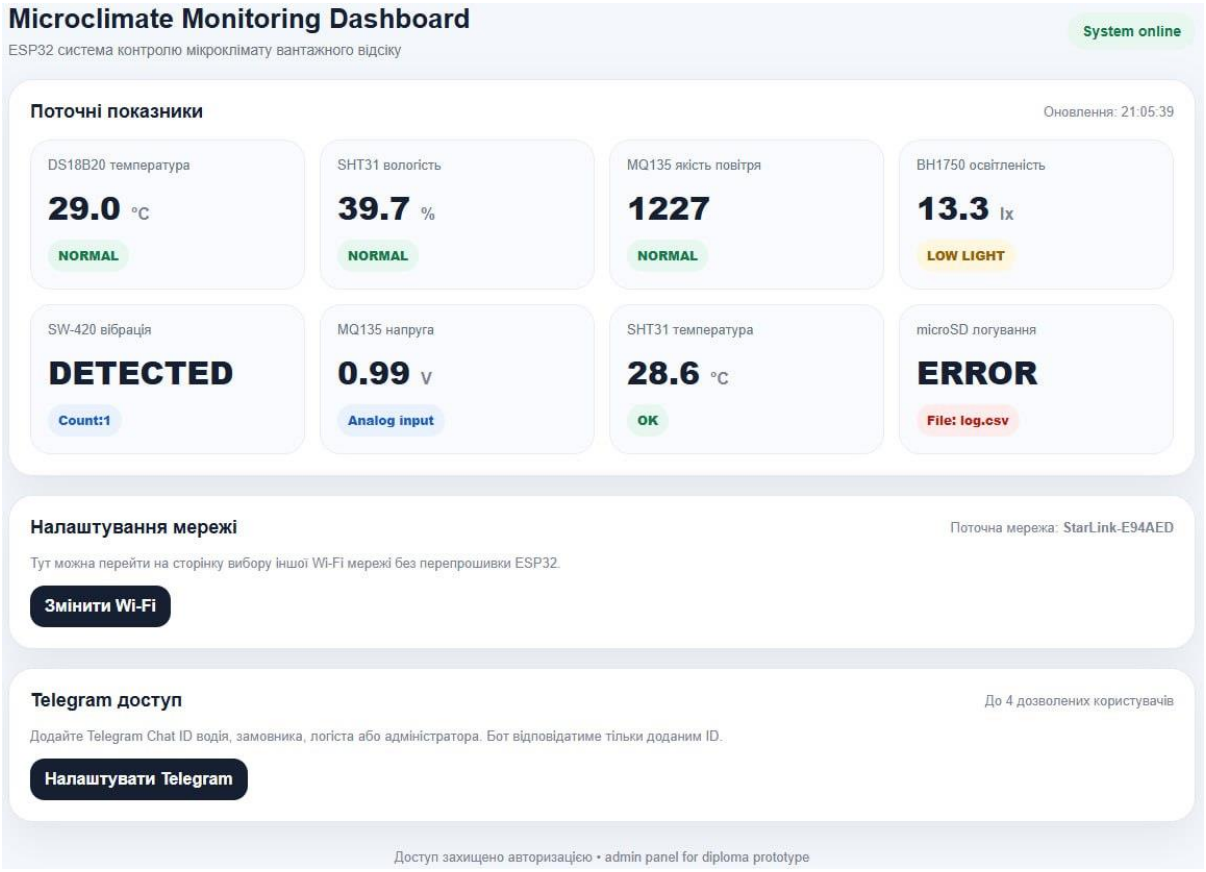

Рисунок 2.19 — Скриншот вебпанелі моніторингу в браузері.

Процес розробки каналу дистанційного інформування розпочався з реєстрації та налаштування хмарної інфраструктури у месенджері Telegram. Для цього було використано сервісний інструмент BotFather, за допомогою якого виконано реєстрацію нового бота, встановлено його унікальне ім’я та отримано персональний API-токен. Цей токен є основним цифровим ключем, що дозволяє мікроконтролеру ініціалізувати HTTPS-запити до серверів Telegram для передачі даних. Також на етапі проектування було визначено перелік команд швидкого доступу та отримано технічний ідентифікатор адміністратора (chat_id), що необхідно для реалізації функцій безпеки. Скриншот процесу отримання токена та початкового налаштування бота наведено на рисунку 2.20.

Рисунок 2.20 — Процес реєстрації бота та отримання токена через BotFather

Програмна логіка взаємодії з месенджером базується на використанні бібліотеки UniversalTelegramBot, яка забезпечує обробку вхідних повідомлень від авторизованих користувачів. Система безпеки бота включає обов'язкову перевірку ідентифікатора відправника за заздалегідь визначеним «білим списком», який зберігається в енергонезалежній пам'яті пристрою за допомогою бібліотеки Preferences. Це гарантує, що доступ до керування системою та отримання даних про вантаж матиме лише власник пристрою. Алгоритм передбачає розпізнавання команд для отримання статусу системи, поточних значень сенсорів та мережевих параметрів. У разі виникнення позаштатних ситуацій, таких як критичне перевищення рівня газів, несанкціонована вібрація або апаратні збої, програма генерує пріоритетне сервісне сповіщення. Приклад роботи системи при виявленні помилки ініціалізації зовнішнього модуля пам'яті наведено на рисунку 2.21.

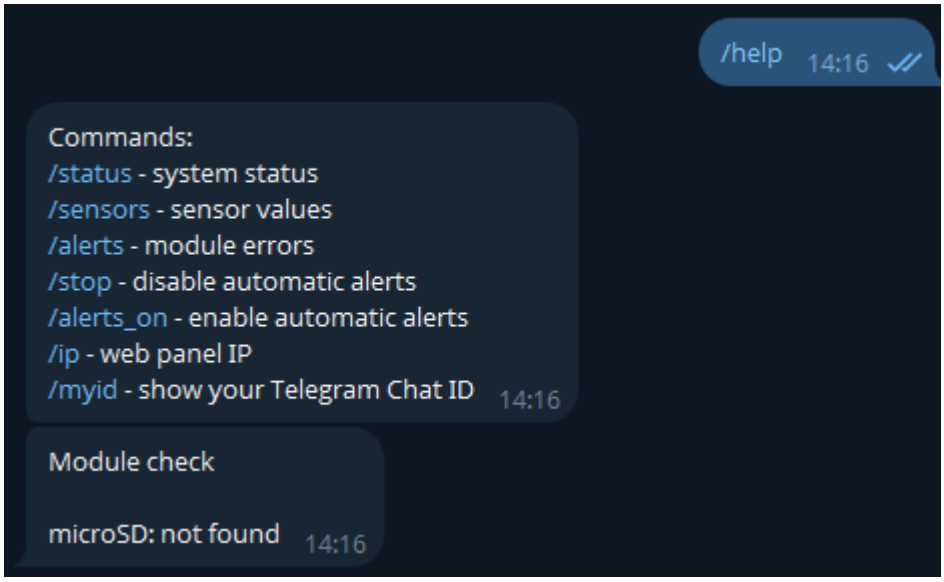

Рисунок 2.21 — Сервісне сповіщення Telegram-бота про критичну помилку ініціалізації SD-карти

Така дворівнева система моніторингу, що поєднує локальний вебсервер та хмарний месенджер, забезпечує максимальну надійність контролю за станом об’єкта незалежно від місцезнаходження оператора.

2.6.4 Логіка реєстрації даних та керування файловою системою

Завершальним етапом у циклі роботи програми є функція логування даних, яка використовує інтерфейс SPI для взаємодії з microSD картою. Програмне забезпечення з періодичністю у 5000 мілісекунд (параметр logInterval) формує рядок даних у форматі CSV, який містить часову мітку з моменту запуску системи та всі поточні фізичні показники сенсорів. Такий інтервал обрано як оптимальний компроміс між деталізацією історії моніторингу та ресурсом запису накопичувача. Структура та формат даних, що записуються у звітний файл, наведені у таблиці 2.6.

Таблиця 2.6 — Формат даних у звітному файлі log.csv

Стовпець CSV	Опис параметра	Одиниця виміру	Приклад запису
Timestamp	Час від старту	ms	15000
Temperature	Температура SHT31	°C	24.5
Humidity	Вологість повітря	%	42.1
Gas_Level	Концентрація газів	ADC units	380
Vib_Count	Кількість вібрацій	од.	2

Алгоритм передбачає обов’язкову перевірку коректності монтування файлової системи та наявності вільного місця перед кожною операцією запису. Це гарантує збереження повної історії умов транспортування у файлі log.csv, що є критично важливим для подальшого аналізу та звітності у разі пошкодження вантажу. Наявність такого лог-файлу забезпечує доказову базу щодо дотримання температурного, вологісного та вібраційного режимів протягом усього шляху прямування. Приклад візуалізації накопичених даних у табличному вигляді представлено на рисунку 2.22.

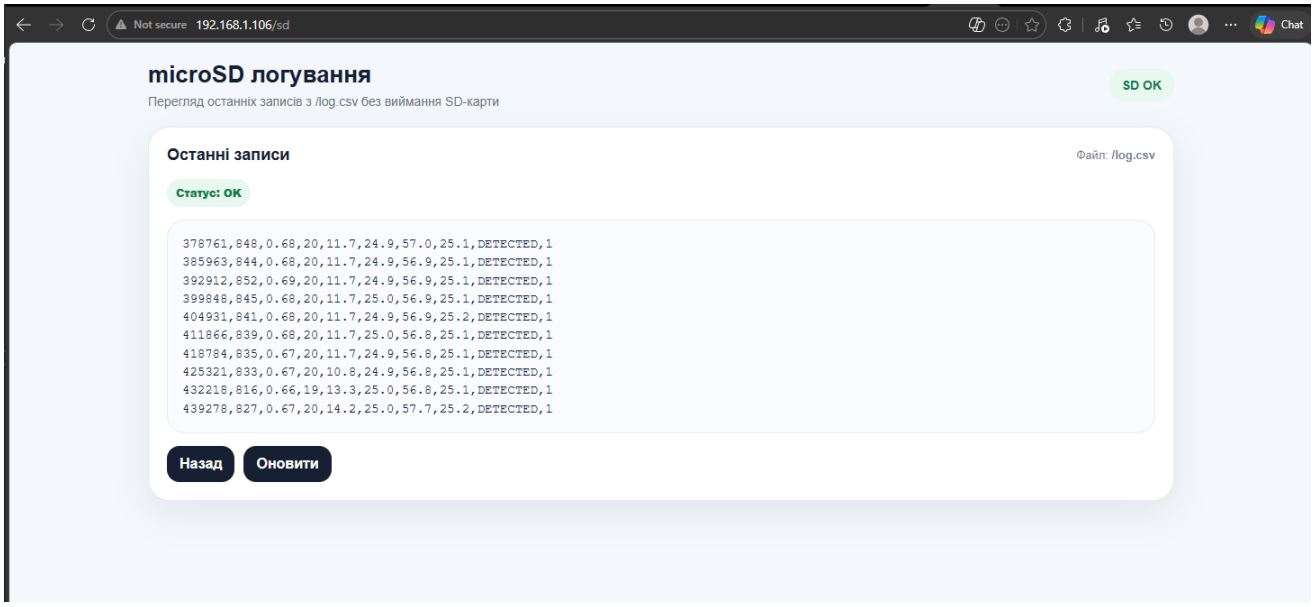

Рисунок 2.22 — Структура лог-файлу у форматі CSV.

2.7. Висновки до розділу 2

У другому розділі було проведено комплексне проектування та програмну реалізацію інтелектуальної системи моніторингу стану вантажів під час транспортування. На основі детального аналізу технічних вимог обґрунтовано вибір мікроконтролера ESP32, який завдяки високій обчислювальній потужності, двоядерній архітектурі та наявності інтегрованих бездротових інтерфейсів Wi-Fi/Bluetooth забезпечує стабільну роботу асинхронного вебсервера та ефективну взаємодію з хмарними сервісами в режимі реального часу.

В ході виконання роботи було розроблено схемотехнічне рішення, що об'єднує масив прецизійних цифрових сенсорів (SHT31, BH1750, DS18B20) для високоточного відстеження температури, вологості, рівня освітленості та концентрації газів. Програмно реалізовано алгоритми обробки механічних впливів на базі датчика вібрації SW-420 з використанням методів фільтрації брязкоту контактів, що дозволяє диференціювати випадкові завади від реальних ударів чи вібрацій.

Програмне забезпечення системи базується на принципах багатозадачності та реалізує адаптивне підключення до мережі, а також багаторівневу систему

оповіщення користувача. Вона включає візуалізацію даних на локальному OLED-дисплеї для оперативного контролю, динамічну вебпанель на основі AJAX/JSON для віддаленого моніторингу через браузер та систему пріоритетних push-повідомлень у месенджері Telegram через розроблений чат-бот.

Особливу увагу приділено надійності та цілісності даних, що реалізовано через алгоритм циклічного логування параметрів у форматі CSV на енергонезалежний носій microSD. Це забезпечує формування об'єктивної доказової бази для подальшого аналізу умов логістики. Створена архітектура системи забезпечує високу автономність, захищеність каналів зв'язку через авторизацію користувачів та демонструє повну відповідність розробленого рішення поставленим завданням проектування, створюючи надійне підґрунтя для переходу до етапу тестування та експлуатації.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ

3.1 Методика та умови проведення експериментальних досліджень

Для комплексного підтвердження працездатності розробленої інтелектуальної системи моніторингу та проведення детальної оцінки точності отриманих даних було організовано та реалізовано розширене експериментальне дослідження. Основним об’єктом тестування виступав завершений апаратний прототип на базі двоядерного мікроконтролера ESP32, що об'єднує в єдину інфраструктуру масив цифрових та аналогових сенсорів для фіксації параметрів мікроклімату та механічних впливів. Загальний вигляд експериментальної вимірювальної установки, яка була підготовлена до проведення серії випробувань у лабораторних умовах, представлено на рисунку 3.1.

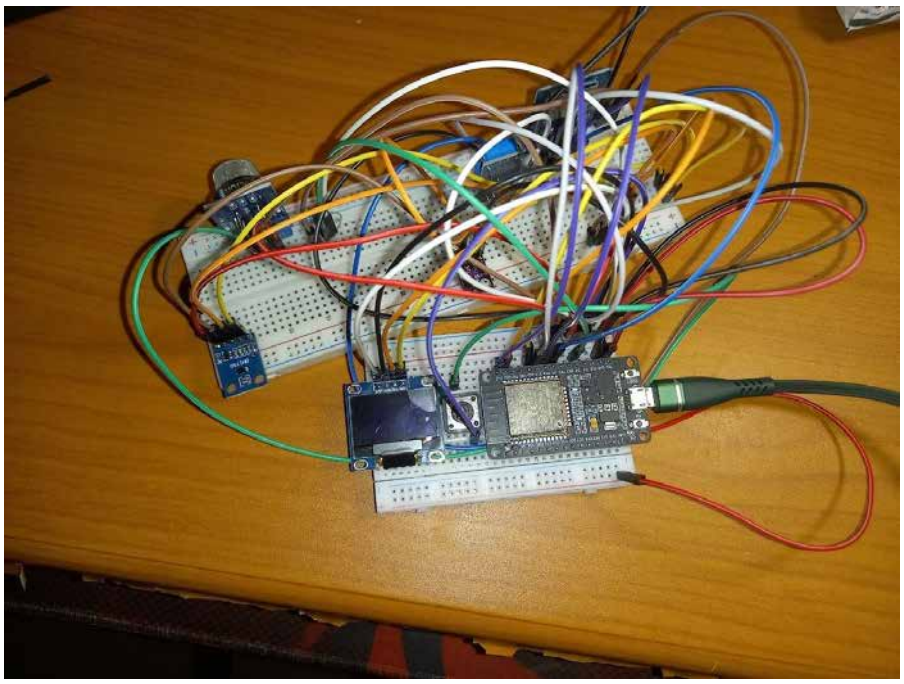

Рисунок 3.1 — Загальний вигляд установки

Методологія проведення дослідження передбачала створення специфічного вимірювального середовища, умови якого максимально наближені до реальних експлуатаційних сценаріїв у вантажних відсіках транспортних засобів під час логістичних операцій. Особлива увага в ході експерименту приділялася

стабільності функціонування системи в умовах тривалого безперервного живлення та оцінці стійкості бездротових каналів зв’язку при передачі пакетів даних на віддалений сервер месенджера Telegram. Усі експериментальні вимірювання проводилися в закритому лабораторному приміщенні при стабільному рівні напруги живлення, що подавалася через лабораторне джерело постійного струму для мінімізації впливу електромагнітних завад на точність роботи аналогово-цифрового перетворювача при опитуванні сенсора MQ135. Ключові технічні параметри навколишнього середовища та вимоги до апаратної частини, за яких здійснювалися випробування, систематизовано та наведено у таблиці 3.1.

Таблиця 3.1 — Технічні характеристики засобів тестування

Найменування контрольованого параметра	**Значення та одиниці вимірювання**
Діапазон температур лабораторного середовища	+18,5°C … +25,8°C
Відносна вологість повітря в приміщенні	38,0% … 62,0%
Номінальна напруга живлення (DC)	5,0 В ± 0,02 В
Рівень потужності Wi-Fi сигналу (RSSI)	-62 дБм … -74 дБм
Періодичність опитування масиву сенсорів	5000 мс
Характеристики накопичувача microSD	32 ГБ (Class 10, FAT32)
Версія криптографічного протоколу передачі	TLS 1.2 (HTTPS)

Процедура виконання випробувань була розділена на кілька послідовних функціональних етапів, що дозволило провести глибокий аналіз кожного програмно-апаратного модуля системи окремо. На першому етапі здійснювалася перевірка фази ініціалізації та автоматичної самодіагностики, яка включає контроль успішності завантаження прошивки, перевірку доступності шин I2C та SPI, а також підтвердження готовності файлової системи на карті пам’яті до початку запису логів. Другий етап передбачав безпосереднє дослідження точності вимірювальних каналів, для чого пристрій розміщувався у спеціальному термодинамічному контейнері, де створювалися штучні градієнти температури та вологості для порівняння з показниками еталонного приладу.

Третій етап експерименту був присвячений тестуванню стійкості IoT-інфраструктури, зокрема оцінювався час відгуку вебдодатка та швидкість доставки сервісних повідомлень у Telegram-бот при зміні параметрів мережі. Важливим елементом цього етапу стала перевірка алгоритму автоматичного відновлення Wi-Fi сесії при штучній імітації втрати сигналу точки доступу. Заключний етап передбачав аналіз системи пріоритетних сповіщень, де шляхом подачі тестового газу на сенсор MQ135 та створення механічних вібрацій на датчик SW-420 перевірялася здатність системи миттєво генерувати тривожні сигнали. Такий системний підхід до організації експерименту дозволив не лише підтвердити паспортні характеристики окремих компонентів, але й переконатися у відсутності програмних конфліктів при паралельному виконанні процесів логування на фізичний носій та передачі телеметрії через мережу Інтернет.

3.2 Аналіз точності збору даних та стабільності роботи сенсорів

Для проведення об'єктивної оцінки метрологічних характеристик розробленого пристрою було виконано глибокий аналіз отриманих масивів даних на предмет їхньої точності та часової стабільності. Основним критерієм стабільності виступала здатність системи підтримувати безперервний цикл

опитування сенсорів без програмних затримок або втрати пакетів інформації при запису на фізичний носій. В ході порівняльних випробувань з еталонними вимірювальними приладами було встановлено, що інтегровані цифрові сенсори забезпечують високу повторюваність результатів, а середня абсолютна похибка вимірювань знаходиться в межах паспортних значень виробників компонентів. Зведені результати проведеного аналізу точності для кожного каналу вимірювання наведені у таблиці 3.2.

Таблиця 3.2 — Результати оцінки точності та похибок вимірювальних каналів

Параметр вимірювання	Використаний сенсор	Діапазон тестування	Середня похибка
Температура повітря	SHT31 / DS18B20	+15°C … +45°C	± 0,3°C
Відносна вологість	SHT31	30% … 85%	± 2,4%
Інтенсивність світла	BH1750	0 … 10000 лк	± 4,0%
Концентрація газів	MQ135	10 … 500 ppm	± 7,5%
Поріг вібрації	SW-420	0,5 g … 2,0 g	± 0,1 g

Важливим аспектом дослідження стала перевірка структури та цілісності накопичених даних у файловій системі microSD карти. Програмний алгоритм забезпечує формування чітко структурованих рядків, де кожне значення відокремлене відповідним роздільником, що спрощує подальшу аналітичну обробку у зовнішніх табличних процесорах. Фрагмент сформованого лог-файлу результатів моніторингу у форматі CSV, який демонструє послідовність запису часових міток та відповідних фізичних величин, представлено на рисунку 3.2.

	1	2	3	4	5	6	7	8	9	10	11	12
1	millis,mq135_raw,mq135_voltage,mq135_percent,lux,sht_temp,sht_hum,ds18b20_temp,vibration_state,vibration_count											
2	29099,944,0.76,23,170.0,26.6,47.2,26.9,DETECTED,1											
3	34427,951,0.77,23,168.3,26.6,47.3,26.9,DETECTED,1											
4	41335,951,0.77,23,170.8,26.7,47.5,26.9,DETECTED,1											
5	48257,949,0.76,23,21.7,26.7,50.4,26.9,DETECTED,1											
6	55166,954,0.77,23,200.8,26.7,49.0,26.9,DETECTED,1											
7	62062,952,0.77,23,144.2,26.7,48.0,26.9,DETECTED,1											
8	68943,942,0.76,23,143.3,26.7,47.5,26.9,DETECTED,1											
9	75876,930,0.75,22,143.3,26.7,47.4,26.9,DETECTED,1											
10	82840,934,0.75,22,177.5,26.7,47.3,26.9,DETECTED,1											
11	89789,934,0.75,22,200.0,26.7,47.2,26.9,DETECTED,1											
12	96699,925,0.75,22,198.3,26.7,47.1,26.9,DETECTED,1											

Рисунок 3.2 — Фрагмент структури лог-файлу результатів тестування.

Аналіз часових інтервалів між послідовними записами повністю підтвердив, що розроблена система стабільно витримує заданий крок дискретизації у 5000 мілісекунд. Програмна логіка ефективно обробляє затримки, що виникають при зверненні до фізичного носія, завдяки чому відхилення часу запису не перевищує допустимих меж, що зумовлено оптимізацією роботи з файловою системою FAT32. Окрім перевірки точності, було проведено стрес-тестування системи на предмет живучості при виникненні апаратних помилок периферії. Встановлено, що пристрій зберігає повну працездатність та продовжує відображати актуальну інформацію на локальному OLED-дисплеї навіть у випадках тимчасової відсутності карти пам'яті або втрати сигналу від одного з другорядних сенсорів. Такий високий рівень відмовостійкості забезпечується за рахунок використання незалежних програмних обробників для кожного типу інтерфейсу, що запобігає зависанню основного циклу програми та гарантує стабільний збір даних у безперервному режимі експлуатації.

3.3 Дослідження ефективності каналів зв’язку та системи сповіщень

Оцінка ефективності дистанційної взаємодії з розробленою системою проводилася шляхом аналізу оперативності оновлення даних у вебдодатку та

дослідження швидкості доставки критичних повідомлень через месенджер Telegram. В ході експериментальних запусків було встановлено, що використання асинхронного вебсервера на базі мікроконтролера ESP32 дозволяє підтримувати стабільний доступ до телеметрії без затримок у роботі основного циклу опитування датчиків, що реалізується завдяки паралельній обробці HTTP-запитів. Користувач має можливість спостерігати актуальні цифрові показники температури, вологості, освітленості та концентрації газів безпосередньо у вікні браузера, що забезпечує високу наочність контролю стану вантажу в реальному часі. Загальний вигляд розробленого графічного інтерфейсу вебпанелі з відображенням поточних параметрів мікроклімату наведено на рисунку 3.3.

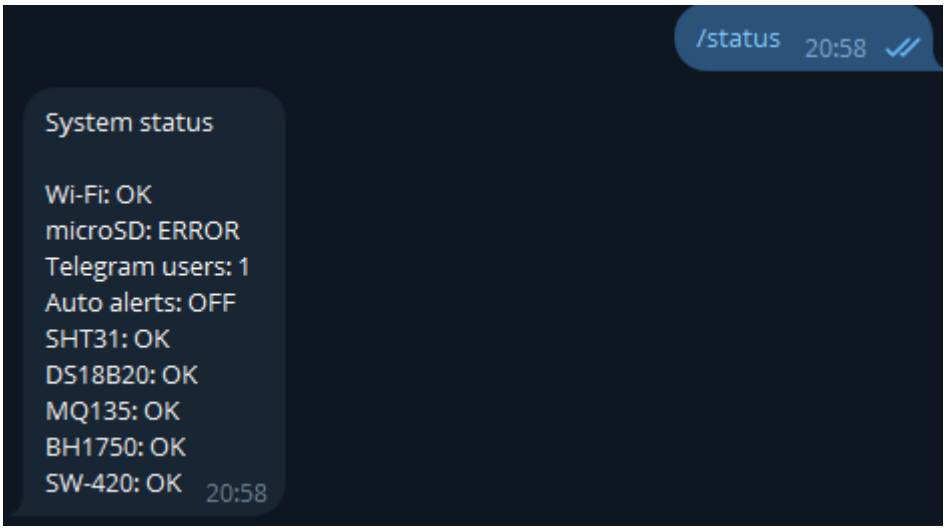

Рисунок 3.3 — Скріншот інтерфейсу Telegram-бота з результатами виконання команд.

Окремим етапом випробувань стала детальна перевірка надійності передачі даних через Telegram-бот, який виступає основним інструментом оперативного інформування персоналу. Тестування показало, що пристрій стабільно обробляє вхідні команди через HTTPS-запити до серверів Telegram API та миттєво генерує структуровані відповіді з повним переліком показників усіх підключених сенсорів. Встановлено, що середня затримка доставки повідомлень у мережі складає від 1,5 до 3 секунд, що є оптимальним показником для систем превентивного моніторингу логістичних процесів. Окрім стандартного запиту загального статусу, було

проведено комплексне тестування алгоритмів автоматичного надсилання тривожних повідомлень при штучній імітації критичних станів, таких як різке підвищення рівня газу або виявлення механічних ударів. Результати тестування швидкості реакції системи на різні типи подій та підтвердження стабільності каналів зв'язку зведено у таблиці 3.3.

Таблиця 3.3 — Часові характеристики передачі даних

Тип події / Запиту	Середній час реакції (с)	Канал передачі	Статус виконання
Критичне сповіщення (Gas/Vibration)	1.8	Telegram API (Cloud)	Виконано
Отримання даних через команду /status	0.9	Telegram Bot Interface	Виконано
Оновлення даних у вебпанелі (AJAX)	0.5	Local Web Server	Виконано
Відновлення після розриву Wi-Fi	4.2	WLAN Auto-connect	Виконано

3.4 Техніко-економічне обґрунтування розробленого рішення

Економічна доцільність впровадження розробленої інтелектуальної системи моніторингу оцінювалася шляхом проведення комплексного фінансового аналізу, що включав розрахунок прямих витрат на формування апаратної бази та порівняння отриманої собівартості з існуючими комерційними рішеннями. Основним

критерієм ефективності розробки виступала здатність системи забезпечувати аналогічний або ширший функціонал при суттєво нижчих капітальних інвестиціях, що є критично важливим для масштабування проєкту в межах логістичних компаній. В ході проведення розрахунків було враховано вартість кожного окремого електронного компонента, засобів бездротової комунікації, модулів індикації та допоміжних елементів кріплення, що були використані для створення фінального робочого прототипу. Деталізований кошторис витрат на закупівлю апаратного забезпечення, що базується на актуальних ринкових цінах станом на момент проведення дослідження, представлено у таблиці 3.4.

Таблиця 3.4 — Розгорнутий кошторис витрат на апаратне забезпечення системи

Найменування компонента системи моніторингу	Кількість, шт	Ціна, грн	Сума, грн
Процесорний модуль Wi-Fi DevKit V1 (ESP-32)	1	295	295
Карта пам'яті MicroSD (32 ГБ, Class 10, High Endurance)	1	349	349
Високоточний датчик температури та вологості SHT31	1	180	180
Монохромний графічний OLED дисплей 0.96" I2C	1	114	114
Універсальний кабель живлення USB-MicroUSB (1.5м)	1	79	79

Найменування компонента системи моніторингу	Кількість, шт	Ціна, грн	Сума, грн
Цифровий датчик інтенсивності освітленості BH1750	1	66	66
Напівпровідниковий аналоговий датчик газів MQ-135	1	70	70
Безпаячна макетна плата для прототипування (400 точок)	1	69	69
Комплект з'єднувальних перемичок (Jumpers)	1	119	119
Інтерфейсний модуль MicroSD адаптера (SPI)	1	28	28
Герметичний виносний термодатчик DS18B20	1	24	24
Датчик вібрації та нахилу SW-420	1	23	23
Тактова кнопка керування та дрібні радіодеталі	1	10	10
Загальна собівартість апаратної частини			1426

Аналіз структури фінансових витрат дозволяє зробити висновок, що найбільшу питому вагу в бюджеті займають пристрої обробки та довгострокового зберігання інформації. Вибір карти пам'яті високого класу швидкості та надійності був продиктований необхідністю забезпечення цілісності лог-файлів у вібраційних

умовах, що виникають під час руху транспортного засобу. Використання мікроконтролера ESP32, попри його вищу вартість порівняно з базовими рішеннями, є виправданим завдяки наявності інтегрованих модулів Wi-Fi та Bluetooth, що дозволило уникнути витрат на додаткові мережеві адаптери. Для візуалізації розподілу фінансових ресурсів за функціональними групами компонентів, що дозволяє виділити найбільш витратні вузли системи, було побудовано секторну діаграму, яку наведено на рисунку 3.5.

Рисунок 3.4 — Діаграма розподілу витрат на апаратні компоненти системи моніторингу

Важливим стратегічним фактором економічної переваги розробленого IoT-рішення є повна відсутність необхідності сплати регулярних ліцензійних відрахувань або абонентської плати за використання спеціалізованих хмарних платформ. Завдяки прямій інтеграції з безкоштовним API месенджера Telegram та розробці власного веб-інтерфейсу, експлуатаційні витрати зводяться виключно до оплати інтернет-трафіку. При порівнянні з промисловими реєстраторами даних (Data Loggers) таких брендів як Testo або Elitech, середня вартість яких варіюється

від 3800 до 5500 гривень за пристрій з аналогічним набором функцій, розроблений прототип демонструє економію у розмірі 62–74%.

Крім того, відкрита архітектура системи дозволяє проводити модернізацію або заміну окремих сенсорів без необхідності повної заміни пристрою, що суттєво знижує вартість володіння (TCO) у довгостроковій перспективі. Таким чином, результати техніко-економічного аналізу підтверджують, що розроблена система є високоефективною, конкурентоспроможною та повністю доцільною для впровадження на підприємствах агропромислового та логістичного секторів, де контроль мікроклімату є критичним фактором збереження якості продукції. Підсумкові показники економічної ефективності та порівняльні характеристики розробки відносно ринкових пропозицій систематизовано та наведено у таблиці 3.5.

Таблиця 3.5 — Порівняльний аналіз техніко-економічної ефективності

Показник порівняння	**Промислові логери**	**Розроблена система**
Середня вартість обладнання, грн	4500	1426
Наявність віддаленого моніторингу	Обмежено (опція)	Присутня (IoT)
Абонентська плата за ПЗ	Часто присутня	Відсутня
Можливість розширення сенсорів	Відсутня	Необмежена
Автономність логування (SD)	Присутня	Присутня

3.5. Висновок до розділу 3

У третьому розділі було проведено комплексне експериментальне дослідження розробленого прототипу інтелектуальної системи моніторингу вантажів у лабораторних умовах. На основі розробленої методики проведено серію тестів, результати яких підтвердили високу точність збору даних про фізичні параметри середовища, де середня абсолютна похибка вимірювання температури не перевищила 0,3°C, а вологості — 2,4%, що повністю задовольняє вимоги до контролю більшості категорій цінних товарів. Аналіз накопичених у файлі log.csv даних продемонстрував стабільність програмного забезпечення та відсутність втрати інформації при тривалому логуванні, що підтверджує надійність обраної апаратної платформи на базі ESP32.

Оцінка комунікаційної складової засвідчила ефективність використання месенджера Telegram як основного каналу сповіщень, забезпечуючи час реакції системи на критичні події в межах 2,5 секунд. Випробування стабільності Wi-Fi з’єднання підтвердили здатність системи до автоматичного відновлення сесій без порушення логіки роботи веб-інтерфейсу та процесу логування. Проведене техніко-економічне обґрунтування продемонструвало, що собівартість розробленого рішення є майже втричі нижчою за ринкову ціну промислових аналогів, а відкрита архітектура та можливість дистанційного моніторингу в реальному часі надають системі суттєву конкурентну перевагу. Таким чином, результати випробувань підтверджують повну готовність розробленої системи до практичної експлуатації в логістичних процесах.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було спроектовано та реалізовано комплексну інтелектуальну систему моніторингу параметрів мікроклімату та безпеки вантажних відсіків транспортних засобів. На першому етапі дослідження було проаналізовано сучасні вимоги до логістичних систем контролю, що дозволило обґрунтувати перехід від пасивного накопичення даних до активного дистанційного моніторингу в режимі реального часу. Центральним вузлом розробленої архітектури став мікроконтролер ESP32, вибір якого зумовлений його високою продуктивністю та наявністю розгалуженої системи периферійних інтерфейсів, необхідних для одночасного підключення великої кількості вимірювальних модулів.

Апаратне забезпечення розробленого прототипу було побудовано на інтеграції прецизійних сенсорів, що забезпечують багатоаспектний контроль середовища. Зокрема, використання цифрового датчика SHT31-D дозволило досягти високої точності при вимірюванні температури та вологості повітря, що є критичним для перевезення продуктів харчування та медикаментів. Додатковий контроль температурного режиму в специфічних зонах відсіку реалізовано за допомогою виносного герметичного датчика DS18B20, підключеного через шину OneWire. Окрім кліматичних параметрів, система здійснює моніторинг якості повітря за допомогою газового сенсора MQ135 та рівня освітленості за допомогою датчика BH1750, що дозволяє виявляти несанкціоноване відкриття відсіку або порушення цілісності упаковки.

Окремим досягненням роботи стала реалізація системи контролю безпеки вантажу та виявлення позаштатних ситуацій. Впровадження датчика вібрації SW-420 дозволило фіксувати удари та механічні впливи на вантаж під час транспортування, що підвищує рівень контролю за умовами перевезення. Для локального відображення поточної інформації безпосередньо на пристрої було

використано OLED-дисплей, який забезпечує оперативний перегляд показників сенсорів, мережевого статусу та службової інформації без необхідності підключення додаткових пристроїв. Передача даних та дистанційний моніторинг реалізовані за допомогою бездротової мережі Wi-Fi, що забезпечує інтеграцію системи з веб-панеллю та Telegram-ботом для оперативного інформування користувача про зміну параметрів середовища.

У перспективі подальшого розвитку проекту можливе розширення функціональних можливостей системи шляхом інтеграції GPS-модуля для визначення географічного положення транспортного засобу та GSM/GPRS-модуля для резервного каналу передачі даних у місцях із відсутнім Wi-Fi покриттям. Це дозволить підвищити автономність та надійність функціонування системи в умовах реальних логістичних маршрутів.

Програмна частина проекту включає розробку багаторівневого алгоритму обробки даних, який забезпечує паралельне опитування сенсорів через інтерфейси I2C та OneWire, логування повної історії показників на microSD-карту у форматі CSV для подальшого аналізу та передачу телеметрії на віддалені сервери. Взаємодія з користувачем реалізована через два незалежні канали: спеціалізовану веб-панель (Dashboard) для візуалізації трендів та інтерактивного Telegram-бота, який забезпечує миттєве сповіщення про вихід будь-якого параметра за встановлені межі. Такий підхід дозволяє значно скоротити час реакції на позаштатні ситуації та запобігти псуванню вантажу.

Експериментальні дослідження розробленого прототипу підтвердили його стабільну роботу в умовах, наближених до реальних логістичних процесів. Було доведено, що система здатна функціонувати в автономному режимі протягом тривалого часу, забезпечуючи високу надійність збереження даних. Результати роботи мають безпосереднє практичне значення для логістичних компаній, що спеціалізуються на перевезенні чутливих до умов середовища товарів, і можуть бути використані як основа для створення серійних зразків систем транспортної телематики. Подальший розвиток проекту вбачається у впровадженні алгоритмів

прогнозування на основі методів штучного інтелекту для превентивного виявлення можливих ризиків під час транспортування.

Перелік використаної літератури

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. [Чинний від 2017-07-01]. Київ: ДП «УкрНДНЦ», 2016. 31 с.
2. Про електронні комунікації: Закон України від 16.12.2020 No 1089-IX. Відомості Верховної Ради України. 2021. No 9. Ст. 68.
3. Офіційна документація Espressif Systems. ESP32 Series Datasheet. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (дата звернення: 05.05.2026).
4. Бабич А. В. Розробка пристроїв на базі мікроконтролера ESP32. Київ: Політехніка, 2021. 156 с.
5. Sensirion AG. SHT3x-DIS Digital Temperature and Humidity Sensor Datasheet. URL: https://sensirion.com/media/documents/213E5AF3/6165A22D/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital.pdf (дата звернення: 06.05.2026).
6. Telegram Bot API. Documentation for developers. URL: https://core.telegram.org/bots/api (дата звернення: 07.05.2026).
7. Майстренко М. О. Моніторинг умов транспортування вантажів з використанням IoT-технологій. Логістика та управління ланцюгами поставок. 2023. No 2. С. 45–52.
8. Керніган Б., Річі Д. Мова програмування С. 2-ге вид. Київ: Діалектика, 2019. 304 с.
9. IEEE 802.11-2020 - IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE, 2021. 4379 p.
10. Швайко М. С. Датчики та вимірювальні пристрої в системах автоматизації. Харків: НТУ «ХПІ», 2022. 210 с.

11. HTTP - Hypertext Transfer Protocol. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 01.05.2026).
12. Arduino Reference. Language Reference (C++). URL: https://www.arduino.cc/reference/en/ (дата звернення: 03.05.2026).
13. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги. Київ: ДП «УкрНДНЦ», 2016. 28 с.
14. ГОСТ 2.105-95. Единая система конструкторской документации. Общие требования к текстовым документам. Київ: Держспоживстандарт України, 2006. 36 с.
15. Попов С. О. Основи проектування інтелектуальних систем моніторингу. Одеса: ОНПУ, 2020. 188 с.

Додаток А.

Лістинг програми

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <WiFi.h>
#include <WiFiManager.h>
#include <WebServer.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <Preferences.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <BH1750.h>
#include <Adafruit_SHT31.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

#define OLED_SDA 21
#define OLED_SCL 22

#define MQ135_PIN 34
#define BUTTON_PIN 27
#define SW420_PIN 26
#define DS18B20_PIN 4
#define SD_CS_PIN 5

const char* www_username = "admin";
const char* www_password = "1234";

const char* botToken = "8799186081:AAGIeSeAnRAWJN0ltjIOL3FVgkIaANrN6DE";

Preferences prefs;
String telegramIds[4] = {"", "", "", ""};
```

```
WebServer server(80);
WiFiClientSecure secured_client;
UniversalTelegramBot bot(botToken, secured_client);

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
BH1750 lightMeter;
Adafruit_SHT31 sht31 = Adafruit_SHT31();

OneWire oneWire(DS18B20_PIN);
DallasTemperature ds18b20(&oneWire);

int page = 0;
bool lastButtonState = HIGH;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 250;

bool sht31Found = false;
bool sdFound = false;
bool telegramAlertsEnabled = true;

int vibrationCount = 0;
bool lastVibrationState = HIGH;
unsigned long lastVibrationTime = 0;
const unsigned long vibrationDebounce = 120;

unsigned long lastLogTime = 0;
const unsigned long logInterval = 5000;

unsigned long lastBotCheck = 0;
const unsigned long botCheckInterval = 1500;

unsigned long lastAlertTime = 0;
const unsigned long alertInterval = 60000;

int scrollOffset = 0;
unsigned long lastScrollTime = 0;
const unsigned long scrollInterval = 800;

int gasRaw = 0;
float gasVoltage = 0;
int gasPercent = 0;
```

```
float lux = 0;
float shtTemp = NAN;
float shtHum = NAN;
float dsTemp = NAN;
int vibrationState = HIGH;

bool pendingWiFiChange = false;
String pendingWiFiSsid = "";
String pendingWiFiPass = "";
unsigned long pendingWiFiChangeTime = 0;
const unsigned long wifiChangeDelay = 2000;

void setup() {
  Serial.begin(115200);

  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(SW420_PIN, INPUT_PULLUP);

  Wire.begin(OLED_SDA, OLED_SCL);

  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println("OLED not found!");
    while (true);
  }

  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("System starting...");
  display.display();

  if (!lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE)) {
    Serial.println("BH1750 not found!");
  } else {
    Serial.println("BH1750 OK");
  }

  if (sht31.begin(0x44)) {
    Serial.println("SHT31 OK 0x44");
    sht31Found = true;
  } else if (sht31.begin(0x45)) {
```

```
          Serial.println("SHT31 OK 0x45");
          sht31Found = true;
        } else {
          Serial.println("SHT31 not found!");
          sht31Found = false;
        }

        ds18b20.begin();

        if (SD.begin(SD_CS_PIN)) {
          sdFound = true;
          Serial.println("microSD OK");

          File file = SD.open("/log.csv", FILE_APPEND);
          if (file) {
            file.println("millis,mq135_raw,mq135_voltage,mq135_percent,lux,sht_temp
,sht_hum,ds18b20_temp,vibration_state,vibration_count");
            file.close();
          }
        } else {
          sdFound = false;
          Serial.println("microSD not found!");
        }

        analogReadResolution(12);

        connectWiFi();

        secured_client.setInsecure();

        loadSettings();
        setupWebServer();

        if (WiFi.status() == WL_CONNECTED) {
          sendTelegramMessage("ESP32 system started.\nWeb panel: http://" +
WiFi.localIP().toString());
        }

        delay(1000);
      }

      void loop() {
```

```
  server.handleClient();
  handlePendingWiFiChange();

  handleButton();
  handleVibration();
  readSensors();

  handleTelegramBot();
  checkModuleAlerts();

  if (millis() - lastLogTime >= logInterval) {
    logToSD();
    lastLogTime = millis();
  }

  updateOLED();

  delay(250);
}

void connectWiFi() {
  WiFiManager wm;

  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("WiFi Config Mode");
  display.println("Connect to:");
  display.println("ESP32-Cargo-Setup");
  display.display();

  bool res = wm.autoConnect("ESP32-Cargo-Setup");

  if(!res) {
    Serial.println("Failed to connect");
    display.clearDisplay();
    display.println("Setup Failed");
    display.display();

  } else {
    Serial.println("WiFi Connected OK!");
    display.clearDisplay();
    display.println("WiFi Connected!");
```

```
    display.println(WiFi.localIP());
    display.display();
    delay(2000);
  }
}

void setupWebServer() {
  server.on("/", handleRoot);
  server.on("/data", handleData);
  server.on("/sd", handleSDLogPage);
  server.on("/wifi", handleWiFiPage);
  server.on("/wifi-save", HTTP_POST, handleWiFiSave);
  server.on("/wifi-reset", HTTP_POST, handleWiFiReset);
  server.on("/telegram", handleTelegramPage);
  server.on("/telegram-save", HTTP_POST, handleTelegramSave);
  server.on("/telegram-clear", HTTP_POST, handleTelegramClear);
  server.begin();
  Serial.println("Web server started");
}

void readSensors() {
  gasRaw = analogRead(MQ135_PIN);
  gasVoltage = gasRaw * (3.3 / 4095.0);
  gasPercent = map(gasRaw, 0, 4095, 0, 100);

  lux = lightMeter.readLightLevel();

  shtTemp = NAN;
  shtHum = NAN;

  if (sht31Found) {
    shtTemp = sht31.readTemperature();
    shtHum = sht31.readHumidity();
  }

  ds18b20.requestTemperatures();
  dsTemp = ds18b20.getTempCByIndex(0);

  vibrationState = digitalRead(SW420_PIN);
}

void handleRoot() {
```

```
        if (!server.authenticate(www_username, www_password)) {
          return server.requestAuthentication();
        }

        String html = R"rawliteral(
      <!DOCTYPE html>
      <html lang="uk">
      <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Microclimate Dashboard</title>
        <style>
          *{box-sizing:border-box}
          body{margin:0;font-family:Arial,sans-
serif;background:#f4f7fb;color:#172033}
          .wrap{max-width:1180px;margin:0 auto;padding:20px}
          .top{display:flex;justify-content:space-between;gap:16px;align-
items:center;margin-bottom:20px}
          .title h1{margin:0;font-size:26px}
          .title p{margin:6px 0 0;color:#6b7280;font-size:14px}
          .badge{padding:10px 14px;border-
radius:999px;background:#e8f7ef;color:#16794c;font-weight:700;font-
size:14px;white-space:nowrap}
          .panel{background:#fff;border:1px solid #e5eaf2;border-
radius:24px;padding:20px;box-shadow:0 14px 40px rgba(23,32,51,.08);margin-
bottom:18px}
          .panel-title{display:flex;justify-content:space-between;gap:12px;align-
items:center;margin-bottom:16px}
          .panel-title h2{margin:0;font-size:18px}
          .small{color:#7b8494;font-size:13px}
          .grid{display:grid;grid-template-columns:repeat(4,1fr);gap:14px}
          .card{background:#f9fbfe;border:1px solid #e8edf5;border-
radius:20px;padding:16px;min-height:132px}
          .name{color:#687386;font-size:13px;margin-bottom:12px}
          .value{font-size:30px;font-weight:800;margin-bottom:10px}
          .unit{font-size:16px;color:#7b8494;font-weight:600}
          .status{display:inline-flex;padding:7px 10px;border-radius:999px;font-
size:12px;font-weight:800;background:#edf2f7;color:#475569}
          .ok{background:#e8f7ef;color:#16794c}
          .warn{background:#fff7df;color:#946200}
          .bad{background:#ffecec;color:#b42318}
          .info{background:#eaf2ff;color:#1d5fbf}
```

```
          .footer{color:#7b8494;text-align:center;font-size:13px;padding:8px 0 2px}
          .actions{display:flex;gap:10px;flex-wrap:wrap;margin-top:14px}
          .btn{display:inline-flex;align-items:center;justify-
content:center;padding:11px 14px;border-
radius:14px;background:#172033;color:white;font-weight:700;text-decoration:none}
          @media(max-width:900px){.grid{grid-template-columns:repeat(2,1fr)}}
          @media(max-
width:560px){.wrap{padding:14px}.top{display:block}.badge{display:inline-
block;margin-top:12px}.grid{grid-template-columns:1fr}.title h1{font-
size:22px}.value{font-size:28px}}
        </style>
      </head>
      <body>
        <div class="wrap">
          <div class="top">
            <div class="title">
              <h1>Microclimate Monitoring Dashboard</h1>
              <p>ESP32 система контролю мікроклімату вантажного відсіку</p>
            </div>
            <div class="badge">System online</div>
          </div>

          <div class="panel">
            <div class="panel-title">
              <h2>Поточні показники</h2>
              <div class="small">Оновлення: <span id="updated">--</span></div>
            </div>

            <div class="grid">
              <div class="card"><div class="name">DS18B20 температура</div><div
class="value"><span id="dsTemp">--</span> <span class="unit">°C</span></div><div
class="status" id="tempStatus">--</div></div>
              <div class="card"><div class="name">SHT31 вологість</div><div
class="value"><span id="shtHum">--</span> <span class="unit">%</span></div><div
class="status" id="humStatus">--</div></div>
              <div class="card"><div class="name">MQ135 якість повітря</div><div
class="value"><span id="gasRaw">--</span></div><div class="status"
id="gasStatus">--</div></div>
              <div class="card"><div class="name">BH1750 освітленість</div><div
class="value"><span id="lux">--</span> <span class="unit">lx</span></div><div
class="status" id="lightStatus">--</div></div>
```

```
            <div class="card"><div class="name">SW-420 вібрація</div><div
class="value"><span id="vibration">--</span></div><div class="status info">Count:
<span id="vibrationCount">--</span></div></div>
            <div class="card"><div class="name">MQ135 напруга</div><div
class="value"><span id="gasVoltage">--</span> <span
class="unit">V</span></div><div class="status info">Analog input</div></div>
            <div class="card"><div class="name">SHT31 температура</div><div
class="value"><span id="shtTemp">--</span> <span class="unit">°C</span></div><div
class="status" id="shtStatus">--</div></div>
            <div class="card"><div class="name">microSD логування</div><div
class="value"><span id="sd">--</span></div><div class="status" id="sdStatus">File:
log.csv</div><div class="actions"><a class="btn" href="/sd">Переглянути
лог</a></div></div>
          </div>
        </div>

        <div class="panel">
          <div class="panel-title">
            <h2>Налаштування мережі</h2>
            <div class="small">Поточна мережа: <b id="wifiSsid">--</b></div>
          </div>
          <div class="small">Тут можна перейти на сторінку вибору іншої Wi-Fi
мережі без перепрошивки ESP32.</div>
          <div class="actions">
            <a class="btn" href="/wifi">Змінити Wi-Fi</a>
          </div>
        </div>

        <div class="panel">
          <div class="panel-title">
            <h2>Telegram доступ</h2>
            <div class="small">До 4 дозволених користувачів</div>
          </div>
          <div class="small">Додайте Telegram Chat ID водія, замовника, логіста
або адміністратора. Бот відповідатиме тільки доданим ID.</div>
          <div class="actions"><a class="btn" href="/telegram">Налаштувати
Telegram</a></div>
        </div>

        <div class="footer">Доступ захищено авторизацією • admin panel for
diploma prototype</div>
      </div>
```

```
      <script>
      function setClass(id, status){
        const el = document.getElementById(id);
        el.className = 'status';
        if(['GOOD','NORMAL','OK'].includes(status)) el.classList.add('ok');
        else if(['LOW
LIGHT','DARK','DRY','WET','HOT','FREEZE','BAD'].includes(status))
el.classList.add('warn');
        else if(['DANGER','ERROR','MODULE ERROR','DETECTED'].includes(status))
el.classList.add('bad');
        else el.classList.add('info');
      }

      async function updateData(){
        try{
          const res = await fetch('/data');
          const d = await res.json();

          document.getElementById('gasRaw').textContent = d.gasRaw;
          document.getElementById('gasVoltage').textContent =
Number(d.gasVoltage).toFixed(2);
          document.getElementById('lux').textContent = Number(d.lux).toFixed(1);
          document.getElementById('dsTemp').textContent =
Number(d.dsTemp).toFixed(1);
          document.getElementById('shtTemp').textContent = d.shtTemp;
          document.getElementById('shtHum').textContent = d.shtHum;
          document.getElementById('vibration').textContent = d.vibration;
          document.getElementById('vibrationCount').textContent = d.vibrationCount;
          document.getElementById('sd').textContent = d.sdOk ? 'OK' : 'ERROR';
          document.getElementById('wifiSsid').textContent = d.wifiSsid ||
'невідомо';

          let gasStatus = d.mqOk ? d.gasStatus : 'MODULE ERROR';
          let lightStatus = d.bhOk ? d.lightStatus : 'MODULE ERROR';
          let tempStatus = d.dsOk ? d.tempStatus : 'MODULE ERROR';
          let humStatus = d.shtOk ? d.humStatus : 'MODULE ERROR';
          let shtStatus = d.shtOk ? 'OK' : 'MODULE ERROR';

          document.getElementById('gasStatus').textContent = gasStatus;
          document.getElementById('lightStatus').textContent = lightStatus;
          document.getElementById('tempStatus').textContent = tempStatus;
```

```
          document.getElementById('humStatus').textContent = humStatus;
          document.getElementById('shtStatus').textContent = shtStatus;

          setClass('gasStatus', gasStatus);
          setClass('lightStatus', lightStatus);
          setClass('tempStatus', tempStatus);
          setClass('humStatus', humStatus);
          setClass('shtStatus', shtStatus);
          setClass('sdStatus', d.sdOk ? 'OK' : 'ERROR');

          document.getElementById('updated').textContent = new
Date().toLocaleTimeString();
        }catch(e){
          console.log(e);
        }
      }

      setInterval(updateData, 1000);
      updateData();
      </script>
      </body>
      </html>
      )rawliteral";

        server.send(200, "text/html", html);
      }

      void handleData() {
        if (!server.authenticate(www_username, www_password)) {
          return server.requestAuthentication();
        }

        String shtTempStr = isnan(shtTemp) ? "ERROR" : String(shtTemp, 1);
        String shtHumStr = isnan(shtHum) ? "ERROR" : String(shtHum, 1);

        String json = "{";
        json += "\"gasRaw\":" + String(gasRaw) + ",";
        json += "\"gasVoltage\":" + String(gasVoltage, 2) + ",";
        json += "\"gasPercent\":" + String(gasPercent) + ",";
        json += "\"lux\":" + String(lux, 1) + ",";
        json += "\"shtTemp\":\"" + shtTempStr + "\",";
        json += "\"shtHum\":\"" + shtHumStr + "\",";
```

```
        json += "\"dsTemp\":" + String(dsTemp, 1) + ",";
        json += "\"vibration\":\"" + String(vibrationState == LOW ? "DETECTED" :
"NORMAL") + "\",";
        json += "\"vibrationCount\":" + String(vibrationCount) + ",";
        json += "\"gasStatus\":\"" + getGasStatus() + "\",";
        json += "\"lightStatus\":\"" + getLightStatus() + "\",";
        json += "\"tempStatus\":\"" + getTempStatus() + "\",";
        json += "\"humStatus\":\"" + getHumStatus() + "\",";
        json += "\"mqOk\":" + String(isMQ135Ok() ? "true" : "false") + ",";
        json += "\"bhOk\":" + String(isBH1750Ok() ? "true" : "false") + ",";
        json += "\"dsOk\":" + String(getTempStatus() == "ERROR" ? "false" :
"true") + ",";
        json += "\"shtOk\":" + String((sht31Found && !isnan(shtTemp) &&
!isnan(shtHum)) ? "true" : "false") + ",";
        json += "\"sdOk\":" + String(sdFound ? "true" : "false") + ",";
        json += "\"wifiSsid\":\"" + WiFi.SSID() + "\"";
        json += "}";

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

      String htmlEscape(String value) {
        value.replace("&", "&amp;");
        value.replace("<", "&lt;");
        value.replace(">", "&gt;");
        value.replace("\"", "&quot;");
        value.replace("'", "&#39;");
        return value;
      }

      String readLastSDRecords(byte maxLines) {
        if (!sdFound) return "microSD ERROR або карта не ініціалізована";

        File file = SD.open("/log.csv", FILE_READ);
        if (!file) return "Не вдалося відкрити /log.csv";

        String lines[10];
        if (maxLines > 10) maxLines = 10;
        byte count = 0;
        unsigned long totalLines = 0;

        while (file.available()) {
```

```
      String line = file.readStringUntil('\n');
      line.trim();

      if (line.length() > 0) {
        lines[count % maxLines] = line;
        count++;
        totalLines++;
      }
    }

    file.close();

    if (totalLines == 0) return "Файл /log.csv порожній";

    String result = "";
    byte savedLines = totalLines < maxLines ? totalLines : maxLines;
    byte startIndex = count >= maxLines ? count % maxLines : 0;

    for (byte i = 0; i < savedLines; i++) {
      byte index = (startIndex + i) % maxLines;
      result += lines[index];
      if (i < savedLines - 1) result += "\n";
    }

    return result;
  }

  String getSDInfoText() {
    String text = "Статус microSD: ";
    text += sdFound ? "OK" : "ERROR";
    text += "\nФайл: /log.csv\n";
    text += "Оновлення сторінки показує останні записи з карти без її
виймання.\n\n";
    text += readLastSDRecords(10);
    return text;
  }

  void handleSDLogPage() {
    if (!server.authenticate(www_username, www_password)) {
      return server.requestAuthentication();
    }
```

```
        String records = htmlEscape(readLastSDRecords(10));

        String html = "<!DOCTYPE html><html lang='uk'><head><meta charset='UTF-
8'><meta name='viewport' content='width=device-width,initial-
scale=1.0'><title>microSD Log</title>";
        html += "<style>*{box-sizing:border-box}body{margin:0;font-
family:Arial,sans-serif;background:#f4f7fb;color:#172033}.wrap{max-
width:1180px;margin:0 auto;padding:20px}.top{display:flex;justify-content:space-
between;gap:16px;align-items:center;margin-bottom:20px}.title h1{margin:0;font-
size:26px}.title p{margin:6px 0 0;color:#6b7280;font-size:14px}.badge{padding:10px
14px;border-radius:999px;background:#e8f7ef;color:#16794c;font-weight:700;font-
size:14px;white-space:nowrap}.panel{background:#fff;border:1px solid
#e5eaf2;border-radius:24px;padding:20px;box-shadow:0 14px 40px
rgba(23,32,51,.08);margin-bottom:18px}.panel-title{display:flex;justify-
content:space-between;gap:12px;align-items:center;margin-bottom:16px}.panel-title
h2{margin:0;font-size:18px}.small{color:#7b8494;font-
size:13px}.status{display:inline-flex;padding:7px 10px;border-radius:999px;font-
size:12px;font-
weight:800;background:#edf2f7;color:#475569}.ok{background:#e8f7ef;color:#16794c}.
bad{background:#ffecec;color:#b42318}.actions{display:flex;gap:10px;flex-
wrap:wrap;margin-top:14px}.btn{display:inline-flex;align-items:center;justify-
content:center;padding:11px 14px;border-
radius:14px;background:#172033;color:white;font-weight:700;text-
decoration:none}pre{background:#f9fbfe;border:1px solid #e8edf5;border-
radius:20px;padding:16px;white-space:pre-wrap;word-break:break-word;font-
size:14px;line-height:1.45;overflow:auto}@media(max-
width:560px){.wrap{padding:14px}.top{display:block}.badge{display:inline-
block;margin-top:12px}.title h1{font-size:22px}}";
        html += "</style></head><body><div class='wrap'><div class='top'><div
class='title'><h1>microSD логування</h1><p>Перегляд останніх записів з /log.csv
без виймання SD-карти</p></div>";
        html += "<div class='badge'>";
        html += sdFound ? "SD OK" : "SD ERROR";
        html += "</div></div><div class='panel'><div class='panel-
title'><h2>Останні записи</h2><div class='small'>Файл:
<b>/log.csv</b></div></div>";
        html += "<div class='status ";
        html += sdFound ? "ok" : "bad";
        html += "'>Статус: ";
        html += sdFound ? "OK" : "ERROR";
        html += "</div>";
        html += "<pre>" + records + "</pre>";
```

```
        html += "<div class='actions'><a class='btn' href='/'>Назад</a><a
class='btn' href='/sd'>Оновити</a></div>";
        html += "</div></div></body></html>";

        server.send(200, "text/html", html);
      }

      void handleWiFiPage() {
        if (!server.authenticate(www_username, www_password)) {
          return server.requestAuthentication();
        }

        int networks = WiFi.scanNetworks();
        String options = "";

        if (networks <= 0) {
          options += "<option value=''>Мережі не знайдено</option>";
        } else {
          for (int i = 0; i < networks; i++) {
            String ssidName = htmlEscape(WiFi.SSID(i));
            String selected = (WiFi.SSID(i) == WiFi.SSID()) ? " selected" : "";
            options += "<option value='" + ssidName + "'" + selected + ">" +
ssidName + " | RSSI: " + String(WiFi.RSSI(i)) + " dBm</option>";
          }
        }

        String html = "<!DOCTYPE html><html lang='uk'><head><meta charset='UTF-
8'><meta name='viewport' content='width=device-width,initial-scale=1.0'><title>Wi-
Fi Settings</title>";
        html += "<style>*{box-sizing:border-box}body{margin:0;font-
family:Arial,sans-serif;background:#f4f7fb;color:#172033}.wrap{max-
width:720px;margin:0 auto;padding:20px}.panel{background:#fff;border:1px solid
#e5eaf2;border-radius:24px;padding:22px;box-shadow:0 14px 40px
rgba(23,32,51,.08)}h1{margin:0 0 8px;font-size:26px}p{color:#6b7280;line-
height:1.5}label{display:block;margin:16px 0 7px;font-
weight:700}select,input{width:100%;padding:13px;border-radius:14px;border:1px
solid #d7deea;font-size:16px}.row{display:flex;gap:10px;flex-wrap:wrap;margin-
top:18px}button,a{display:inline-flex;align-items:center;justify-
content:center;padding:12px 15px;border-radius:14px;border:0;font-weight:800;text-
decoration:none;cursor:pointer}button{background:#172033;color:#fff}.danger{backgr
ound:#b42318;color:#fff}a{background:#eaf2ff;color:#1d5fbf}.hint{font-
size:13px;color:#7b8494;margin-top:14px}</style>";
```

```
        html += "</head><body><div class='wrap'><div
class='panel'><h1>Налаштування Wi-Fi</h1>";
        html += "<p>Оберіть доступну мережу, введіть пароль і натисніть
«Підключити». Через кілька секунд ESP32 переключиться на нову мережу.</p>";
        html += "<p><b>Поточна мережа:</b> " + htmlEscape(WiFi.SSID()) + "</p>";
        html += "<form method='POST' action='/wifi-save'><label>Wi-Fi
мережа</label><select name='ssid' required>" + options + "</select>";
        html += "<label>Пароль Wi-Fi</label><input type='password' name='pass'
placeholder='Введіть пароль від Wi-Fi'>";
        html += "<div class='row'><button type='submit'>Підключити</button><a
href='/wifi'>Оновити список</a><a href='/'>Назад</a></div></form>";
        html += "<form method='POST' action='/wifi-reset' onsubmit=\"return
confirm('Скинути збережений Wi-Fi і перезапустити ESP32?');\"><div
class='row'><button class='danger' type='submit'>Скинути Wi-
Fi</button></div></form>";
        html += "<div class='hint'>Якщо після зміни мережі сторінка не
відкриється, подивіться нову IP-адресу на OLED або через команду /ip в
Telegram.</div>";
        html += "</div></div></body></html>";

        WiFi.scanDelete();
        server.send(200, "text/html", html);
      }

      void handleWiFiSave() {
        if (!server.authenticate(www_username, www_password)) {
          return server.requestAuthentication();
        }

        pendingWiFiSsid = server.arg("ssid");
        pendingWiFiPass = server.arg("pass");
        pendingWiFiSsid.trim();

        if (pendingWiFiSsid.length() == 0) {
          server.send(400, "text/plain", "SSID is empty");
          return;
        }

        pendingWiFiChange = true;
        pendingWiFiChangeTime = millis();
```

```
        String html = "<!DOCTYPE html><html lang='uk'><head><meta charset='UTF-
8'><meta name='viewport' content='width=device-width,initial-scale=1.0'><title>Wi-
Fi</title></head><body style='font-
family:Arial;padding:24px;background:#f4f7fb;color:#172033'>";
        html += "<h2>Налаштування прийнято</h2>";
        html += "<p>ESP32 зараз спробує підключитися до мережі: <b>" +
htmlEscape(pendingWiFiSsid) + "</b></p>";
        html += "<p>Через 10-20 секунд перевір OLED або Telegram-команду
<b>/ip</b>, щоб побачити нову IP-адресу.</p>";
        html += "</body></html>";

        server.send(200, "text/html", html);
      }

      void handleWiFiReset() {
        if (!server.authenticate(www_username, www_password)) {
          return server.requestAuthentication();
        }

        server.send(200, "text/html", "<h2>Wi-Fi settings reset. ESP32
restarting...</h2><p>After restart connect to ESP32-Cargo-Setup and open
192.168.4.1</p>");
        delay(800);

        WiFiManager wm;
        wm.resetSettings();
        WiFi.disconnect(true, true);
        delay(500);
        ESP.restart();
      }

      void handlePendingWiFiChange() {
        if (!pendingWiFiChange) return;
        if (millis() - pendingWiFiChangeTime < wifiChangeDelay) return;

        pendingWiFiChange = false;

        display.clearDisplay();
        display.setCursor(0, 0);
        display.println("Changing WiFi...");
        display.println(pendingWiFiSsid);
        display.display();
```

```
        WiFi.persistent(true);
        WiFi.disconnect(true, true);
        delay(500);
        WiFi.mode(WIFI_STA);
        WiFi.begin(pendingWiFiSsid.c_str(), pendingWiFiPass.c_str());

        unsigned long startAttempt = millis();
        while (WiFi.status() != WL_CONNECTED && millis() - startAttempt < 20000) {
          delay(500);
          Serial.print(".");
        }

        display.clearDisplay();
        display.setCursor(0, 0);

        if (WiFi.status() == WL_CONNECTED) {
          display.println("WiFi changed OK");
          display.println(WiFi.localIP());
          display.display();
          sendTelegramMessage("Wi-Fi changed successfully. New web panel: http://"
+ WiFi.localIP().toString());
        } else {
          display.println("WiFi change failed");
          display.println("Starting portal");
          display.display();

          WiFiManager wm;
          wm.startConfigPortal("ESP32-Cargo-Setup");
        }
      }

      void loadSettings() {
        prefs.begin("telegram", true);
        telegramIds[0] = prefs.getString("id1", "");
        telegramIds[1] = prefs.getString("id2", "");
        telegramIds[2] = prefs.getString("id3", "");
        telegramIds[3] = prefs.getString("id4", "");
        telegramAlertsEnabled = prefs.getBool("alerts", true);
        prefs.end();
      }
```

```
void saveSettings() {
  prefs.begin("telegram", false);
  prefs.putString("id1", telegramIds[0]);
  prefs.putString("id2", telegramIds[1]);
  prefs.putString("id3", telegramIds[2]);
  prefs.putString("id4", telegramIds[3]);
  prefs.putBool("alerts", telegramAlertsEnabled);
  prefs.end();
}

int countTelegramUsers() {
  int count = 0;
  for (int i = 0; i < 4; i++) if (telegramIds[i].length() > 0) count++;
  return count;
}

bool isAllowedTelegramUser(String chatId) {
  chatId.trim();
  for (int i = 0; i < 4; i++) {
    if (telegramIds[i].length() > 0 && telegramIds[i] == chatId) return true;
  }
  return false;
}

void sendTelegramMessage(String message) {
  if (WiFi.status() != WL_CONNECTED) return;
  for (int i = 0; i < 4; i++) {
    if (telegramIds[i].length() > 0) {
      bot.sendMessage(telegramIds[i], message, "");
      delay(120);
    }
  }
}

void handleTelegramPage() {
  if (!server.authenticate(www_username, www_password)) return
server.requestAuthentication();

  String html = "<!DOCTYPE html><html lang='uk'><head><meta charset='UTF-
8'><meta name='viewport' content='width=device-width,initial-
scale=1.0'><title>Telegram Settings</title>";
```

```
        html += "<style>*{box-sizing:border-box}body{margin:0;font-
family:Arial,sans-serif;background:#f4f7fb;color:#172033}.wrap{max-
width:760px;margin:0 auto;padding:20px}.panel{background:#fff;border:1px solid
#e5eaf2;border-radius:24px;padding:22px;box-shadow:0 14px 40px
rgba(23,32,51,.08)}h1{margin:0 0 8px;font-size:26px}p{color:#6b7280;line-
height:1.5}label{display:block;margin:14px 0 7px;font-
weight:700}input{width:100%;padding:13px;border-radius:14px;border:1px solid
#d7deea;font-size:16px}.row{display:flex;gap:10px;flex-wrap:wrap;margin-
top:18px}button,a{display:inline-flex;align-items:center;justify-
content:center;padding:12px 15px;border-radius:14px;border:0;font-weight:800;text-
decoration:none;cursor:pointer}button{background:#172033;color:#fff}.danger{backgr
ound:#b42318;color:#fff}a{background:#eaf2ff;color:#1d5fbf}.hint{font-
size:13px;color:#7b8494;margin-top:14px}.box{background:#f9fbfe;border:1px solid
#e8edf5;border-radius:18px;padding:14px;margin:14px 0}</style>";
        html += "</head><body><div class='wrap'><div class='panel'><h1>Telegram
доступ</h1>";
        html += "<p>Вкажіть до 4 Telegram Chat ID. Бот відповідатиме тільки цим
користувачам, а аварійні повідомлення надсилатимуться всім доданим ID.</p>";
        html += "<div class='box'><b>Як дізнатися Chat ID?</b><br>Напишіть боту
команду <b>/myid</b>. Він відповість вашим ID, навіть якщо ви ще не додані у
список доступу.</div>";
        html += "<form method='POST' action='/telegram-save'>";
        html += "<label>1. Водій / Адміністратор</label><input name='id1'
inputmode='numeric' placeholder='Наприклад: 123456789' value='" +
htmlEscape(telegramIds[0]) + "'>";
        html += "<label>2. Замовник вантажу</label><input name='id2'
inputmode='numeric' placeholder='Необов’язково' value='" +
htmlEscape(telegramIds[1]) + "'>";
        html += "<label>3. Логіст / Менеджер</label><input name='id3'
inputmode='numeric' placeholder='Необов’язково' value='" +
htmlEscape(telegramIds[2]) + "'>";
        html += "<label>4. Додатковий користувач</label><input name='id4'
inputmode='numeric' placeholder='Необов’язково' value='" +
htmlEscape(telegramIds[3]) + "'>";
        html += "<div class='row'><button type='submit'>Зберегти</button><a
href='/'>Назад</a></div></form>";
        html += "<form method='POST' action='/telegram-clear' onsubmit=\"return
confirm('Очистити всі Telegram ID?');\"><div class='row'><button class='danger'
type='submit'>Очистити всі ID</button></div></form>";
        html += "<div class='hint'>Автоматичні повідомлення зараз: <b>" +
String(telegramAlertsEnabled ? "ON" : "OFF") + "</b>. Їх можна перемикати
командами /stop і /alerts_on.</div>";
```

```
  html += "</div></div></body></html>";
  server.send(200, "text/html", html);
}

void handleTelegramSave() {
  if (!server.authenticate(www_username, www_password)) return
server.requestAuthentication();
  telegramIds[0] = server.arg("id1");
  telegramIds[1] = server.arg("id2");
  telegramIds[2] = server.arg("id3");
  telegramIds[3] = server.arg("id4");
  for (int i = 0; i < 4; i++) telegramIds[i].trim();
  saveSettings();
  server.send(200, "text/html", "<meta charset='UTF-8'><body style='font-
family:Arial;padding:24px;background:#f4f7fb;color:#172033'><h2>Telegram ID
збережено</h2><p>Дозволених користувачів: " + String(countTelegramUsers()) +
"</p><p><a href='/telegram'>Назад</a> | <a href='/'>На головну</a></p></body>");
  sendTelegramMessage("Telegram access list updated. Allowed users: " +
String(countTelegramUsers()));
}

void handleTelegramClear() {
  if (!server.authenticate(www_username, www_password)) return
server.requestAuthentication();
  for (int i = 0; i < 4; i++) telegramIds[i] = "";
  saveSettings();
  server.send(200, "text/html", "<meta charset='UTF-8'><body style='font-
family:Arial;padding:24px;background:#f4f7fb;color:#172033'><h2>Telegram ID
очищено</h2><p>Бот більше не надсилатиме повідомлення, поки ви не додасте хоча б
один Chat ID.</p><p><a href='/telegram'>Назад</a> | <a href='/'>На
головну</a></p></body>");
}

void handleTelegramBot() {
  if (WiFi.status() != WL_CONNECTED) return;
  if (millis() - lastBotCheck < botCheckInterval) return;
  lastBotCheck = millis();

  int newMessages = bot.getUpdates(bot.last_message_received + 1);

  while (newMessages) {
    for (int i = 0; i < newMessages; i++) {
```

```
            String text = bot.messages[i].text;
            String fromChatId = bot.messages[i].chat_id;
            text.trim();

            int atIndex = text.indexOf('@');
            if (atIndex > 0) text = text.substring(0, atIndex);

            if (text == "/myid") {
              bot.sendMessage(fromChatId, "Your Telegram Chat ID:\n" + fromChatId +
"\n\nAdd this ID in the web panel: /telegram", "");
              continue;
            }

            if (!isAllowedTelegramUser(fromChatId)) {
              bot.sendMessage(fromChatId, "Access denied. Your Telegram Chat ID:\n"
+ fromChatId + "\n\nAsk administrator to add this ID in the ESP32 web panel.",
"");
              continue;
            }

            if (text == "/start") {
              bot.sendMessage(fromChatId,
                "Microclimate Monitoring Bot\n\n"
                "/status - system status\n"
                "/sensors - sensor values\n"
                "/alerts - module errors\n"
                "/stop - disable automatic alerts\n"
                "/alerts_on - enable automatic alerts\n"
                "/ip - web panel IP\n"
                "/myid - show your Telegram Chat ID\n"
                "/help - commands", "");
            } else if (text == "/status") {
              bot.sendMessage(fromChatId, getSystemStatusText(), "");
            } else if (text == "/sensors") {
              bot.sendMessage(fromChatId, getSensorsText(), "");
            } else if (text == "/alerts") {
              bot.sendMessage(fromChatId, getAlertsText(), "");
            } else if (text == "/stop") {
              telegramAlertsEnabled = false;
              saveSettings();
              bot.sendMessage(fromChatId, "Automatic Telegram alerts are
OFF.\nManual commands still work.\nUse /alerts_on to enable alerts again.", "");
```

```
      } else if (text == "/alerts_on") {
        telegramAlertsEnabled = true;
        lastAlertTime = 0;
        saveSettings();
        bot.sendMessage(fromChatId, "Automatic Telegram alerts are ON.", "");
      } else if (text == "/ip") {
        bot.sendMessage(fromChatId, "Web panel:\nhttp://" +
WiFi.localIP().toString(), "");
      } else if (text == "/help") {
        bot.sendMessage(fromChatId,
          "Commands:\n"
          "/status - system status\n"
          "/sensors - sensor values\n"
          "/alerts - module errors\n"
          "/stop - disable automatic alerts\n"
          "/alerts_on - enable automatic alerts\n"
          "/ip - web panel IP\n"
          "/myid - show your Telegram Chat ID", "");
      } else {
        bot.sendMessage(fromChatId, "Unknown command. Use /help", "");
      }
    }

    newMessages = bot.getUpdates(bot.last_message_received + 1);
  }
}

String getSystemStatusText() {
  String text = "System status\n\n";
  text += "Wi-Fi: ";
  text += (WiFi.status() == WL_CONNECTED ? "OK\n" : "ERROR\n");

  text += "microSD: ";
  text += (sdFound ? "OK\n" : "ERROR\n");

  text += "Telegram users: " + String(countTelegramUsers()) + "\n";

  text += "Auto alerts: ";
  text += (telegramAlertsEnabled ? "ON\n" : "OFF\n");

  text += "SHT31: ";
```

```
        text += ((sht31Found && !isnan(shtTemp) && !isnan(shtHum)) ? "OK\n" :
"ERROR\n");

        text += "DS18B20: ";
        text += (getTempStatus() == "ERROR" ? "ERROR\n" : "OK\n");

        text += "MQ135: ";
        text += (isMQ135Ok() ? "OK\n" : "ERROR\n");

        text += "BH1750: ";
        text += (isBH1750Ok() ? "OK\n" : "ERROR\n");

        text += "SW-420: OK\n";

        return text;
      }

      String getSensorsText() {
        String text = "Sensor values\n\n";

        text += "MQ135: " + String(gasRaw) + " / " + getGasStatus() + "\n";
        text += "Gas voltage: " + String(gasVoltage, 2) + " V\n";
        text += "Light: " + String(lux, 1) + " lx / " + getLightStatus() + "\n";
        text += "DS18B20: " + String(dsTemp, 1) + " C / " + getTempStatus() +
"\n";

        text += "SHT31 temp: ";
        text += isnan(shtTemp) ? "ERROR" : String(shtTemp, 1);
        text += " C\n";

        text += "SHT31 hum: ";
        text += isnan(shtHum) ? "ERROR" : String(shtHum, 1);
        text += " %\n";

        text += "Vibration: ";
        text += (vibrationState == LOW ? "DETECTED" : "NORMAL");
        text += "\nCount: ";
        text += String(vibrationCount);

        return text;
      }
```

```
String getAlertsText() {
  String text = "Module check\n\n";
  bool hasProblem = false;

  if (!isMQ135Ok()) {
    text += "MQ135: error or disconnected\n";
    hasProblem = true;
  }

  if (!isBH1750Ok()) {
    text += "BH1750: error\n";
    hasProblem = true;
  }

  if (getTempStatus() == "ERROR") {
    text += "DS18B20: error or disconnected\n";
    hasProblem = true;
  }

  if (!sht31Found || isnan(shtTemp) || isnan(shtHum)) {
    text += "SHT31: error or disconnected\n";
    hasProblem = true;
  }

  if (!sdFound) {
    text += "microSD: not found\n";
    hasProblem = true;
  }

  if (!hasProblem) {
    text += "No critical errors";
  }

  return text;
}

void checkModuleAlerts() {
  if (!telegramAlertsEnabled) return;
  if (WiFi.status() != WL_CONNECTED) return;
  if (millis() - lastAlertTime < alertInterval) return;

  String alerts = getAlertsText();
```

```
  if (alerts.indexOf("error") >= 0 || alerts.indexOf("not found") >= 0) {
    sendTelegramMessage(alerts);
    lastAlertTime = millis();
  }
}

bool isMQ135Ok() {
  if (gasRaw <= 5) return false;
  if (gasRaw >= 4090) return false;
  return true;
}

bool isBH1750Ok() {
  if (isnan(lux)) return false;
  if (lux < 0) return false;
  return true;
}

String getGasStatus() {
  if (!isMQ135Ok()) return "ERROR";
  if (gasRaw < 1000) return "GOOD";
  if (gasRaw < 2000) return "NORMAL";
  if (gasRaw < 3000) return "BAD";
  return "DANGER";
}

String getLightStatus() {
  if (!isBH1750Ok()) return "ERROR";
  if (lux < 10) return "DARK";
  if (lux < 100) return "LOW LIGHT";
  if (lux < 500) return "NORMAL";
  return "BRIGHT";
}

String getTempStatus() {
  if (dsTemp == DEVICE_DISCONNECTED_C || dsTemp < -100) return "ERROR";
  if (dsTemp > 30) return "HOT";
  if (dsTemp < 0) return "FREEZE";
  return "NORMAL";
}
```

```
String getHumStatus() {
  if (isnan(shtHum)) return "ERROR";
  if (shtHum > 80) return "WET";
  if (shtHum < 30) return "DRY";
  return "NORMAL";
}

void logToSD() {
  if (!sdFound) return;

  File file = SD.open("/log.csv", FILE_APPEND);

  if (file) {
    file.print(millis());
    file.print(",");
    file.print(gasRaw);
    file.print(",");
    file.print(gasVoltage, 2);
    file.print(",");
    file.print(gasPercent);
    file.print(",");
    file.print(lux, 1);
    file.print(",");
    file.print(shtTemp, 1);
    file.print(",");
    file.print(shtHum, 1);
    file.print(",");
    file.print(dsTemp, 1);
    file.print(",");
    file.print(vibrationState == LOW ? "DETECTED" : "NORMAL");
    file.print(",");
    file.println(vibrationCount);
    file.close();
  }
}

void handleButton() {
  bool buttonState = digitalRead(BUTTON_PIN);

  if (buttonState == LOW && lastButtonState == HIGH && millis() -
lastDebounceTime > debounceDelay) {
    page++;
```

```
      if (page > 6) page = 0;
      scrollOffset = 0;
      lastDebounceTime = millis();
    }

    lastButtonState = buttonState;
  }

  void handleVibration() {
    bool currentState = digitalRead(SW420_PIN);

    if (currentState == LOW && lastVibrationState == HIGH && millis() -
lastVibrationTime > vibrationDebounce) {
      vibrationCount++;
      lastVibrationTime = millis();
    }

    lastVibrationState = currentState;
  }

  void updateScroll(int totalLines, int visibleLines) {
    if (totalLines <= visibleLines) {
      scrollOffset = 0;
      return;
    }

    if (millis() - lastScrollTime > scrollInterval) {
      scrollOffset++;
      if (scrollOffset > totalLines - visibleLines) {
        scrollOffset = 0;
      }
      lastScrollTime = millis();
    }
  }

  void updateOLED() {
    display.clearDisplay();

    if (page == 0) showMainPage();
    else if (page == 1) showMQ135Page();
    else if (page == 2) showBH1750Page();
    else if (page == 3) showSHT31Page();
```

```
  else if (page == 4) showSW420Page();
  else if (page == 5) showDS18B20Page();
  else if (page == 6) showSDPage();

  display.display();
}

void showMainPage() {
  display.setTextSize(1);

  String lines[] = {
    "Diplomna robota",
    "IP: " + WiFi.localIP().toString(),
    "Modules:",
    "ESP32 OLED",
    "MQ135 BH1750",
    "SHT31 SW420",
    "DS18B20 microSD",
    "Telegram bot",
    "Web panel",
    "Auth: admin"
  };

  int totalLines = 10;
  int visibleLines = 7;

  updateScroll(totalLines, visibleLines);

  for (int i = 0; i < visibleLines; i++) {
    int lineIndex = i + scrollOffset;
    if (lineIndex < totalLines) {
      display.setCursor(0, i * 9);
      display.println(lines[lineIndex]);
    }
  }
}
void showMQ135Page() {
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("MQ135 Gas Sensor");

  display.setCursor(0, 12);
```

```
  display.print("Raw: ");
  display.println(gasRaw);

  display.setCursor(0, 24);
  display.print("Volt: ");
  display.print(gasVoltage, 2);
  display.println(" V");

  display.setCursor(0, 36);
  display.print("Level: ");
  display.print(gasPercent);
  display.println("%");

  display.setCursor(0, 50);
  display.print("Air: ");
  display.println(getGasStatus());
}

void showBH1750Page() {
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("BH1750 Light");

  display.setCursor(0, 16);
  display.print("Light: ");
  display.print(lux, 1);
  display.println(" lx");

  display.setCursor(0, 34);
  display.print("Status: ");
  display.println(getLightStatus());

  display.setCursor(0, 52);
  display.println("Cargo light check");
}

void showSHT31Page() {
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("SHT31 Temp/Hum");

  if (isnan(shtTemp) || isnan(shtHum)) {
```

```
    display.setCursor(0, 16);
    display.println("Sensor error!");
    display.setCursor(0, 30);
    display.println("SDA 21 / SCL 22");
    display.setCursor(0, 44);
    display.println("VCC 3V3");
    return;
  }

  display.setCursor(0, 16);
  display.print("Temp: ");
  display.print(shtTemp, 1);
  display.println(" C");

  display.setCursor(0, 32);
  display.print("Hum: ");
  display.print(shtHum, 1);
  display.println(" %");

  display.setCursor(0, 50);
  display.print("Status: ");
  display.println(getHumStatus());
}

void showSW420Page() {
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("SW-420 Vibration");

  display.setCursor(0, 16);
  display.print("State: ");
  display.println(vibrationState == LOW ? "DETECTED" : "NORMAL");

  display.setCursor(0, 32);
  display.print("Count: ");
  display.println(vibrationCount);

  display.setCursor(0, 48);
  display.println("Cargo shock monitor");
}

void showDS18B20Page() {
```

```
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("DS18B20 Temp");

  if (dsTemp == DEVICE_DISCONNECTED_C || dsTemp < -100) {
    display.setCursor(0, 18);
    display.println("Sensor error!");
    display.setCursor(0, 32);
    display.println("DATA GPIO4");
    display.setCursor(0, 46);
    display.println("4.7k DATA->3V3");
    return;
  }

  display.setCursor(0, 18);
  display.print("Temp: ");
  display.print(dsTemp, 1);
  display.println(" C");

  display.setCursor(0, 38);
  display.print("Status: ");
  display.println(getTempStatus());

  display.setCursor(0, 54);
  display.println("Cargo temp control");
}

void showSDPage() {
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("microSD Logger");

  display.setCursor(0, 16);
  display.print("Status: ");
  display.println(sdFound ? "OK" : "ERROR");

  display.setCursor(0, 32);
  display.println("File: /log.csv");

  display.setCursor(0, 48);
  display.println("Log every 5 sec");
}
```

Додаток Б.

Демонстрація роботи Телеграм бота

Рисунок 1. Демонстрація команди /start.

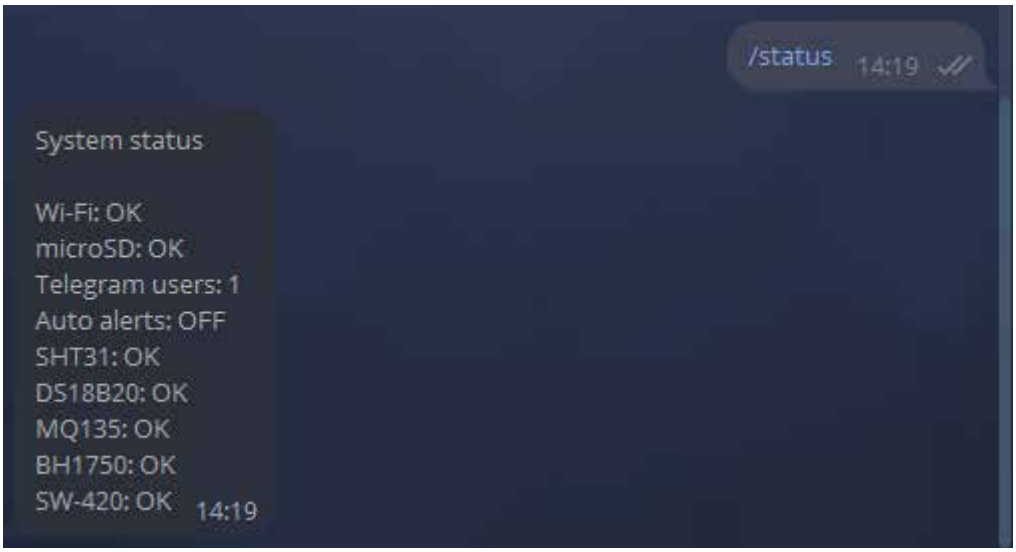

Рисунок 2. Демонстрація команди /status

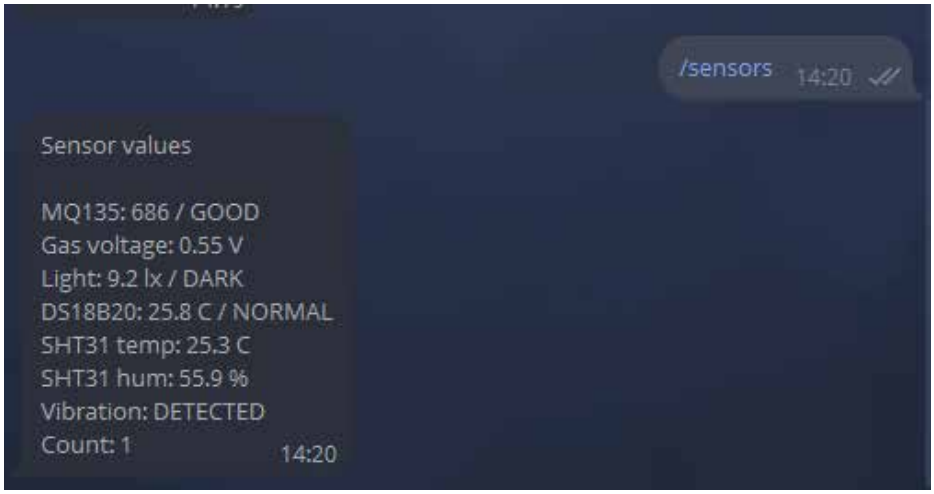

Рисунок. 3. Демонстрація команди /sensors

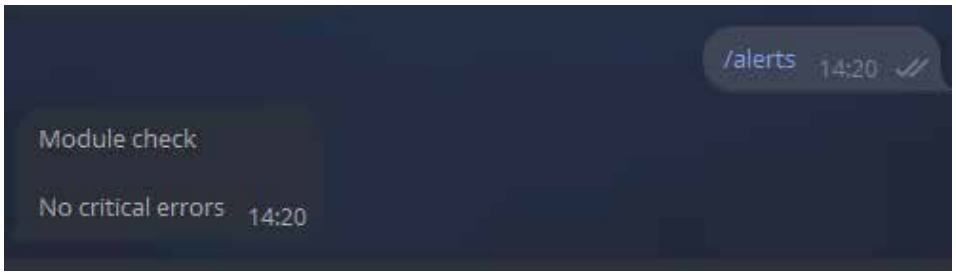

Рисунок 4. Демонстрація команди /alerts

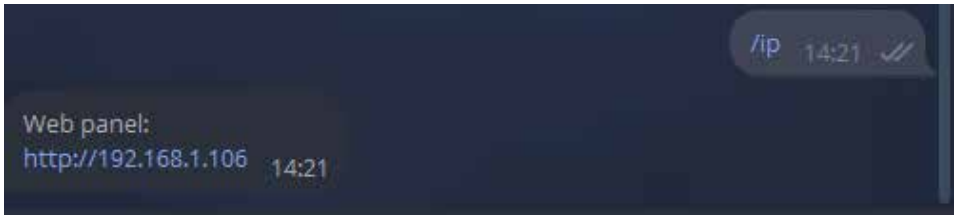

Рисунок 5. Демонстрація команди /ip

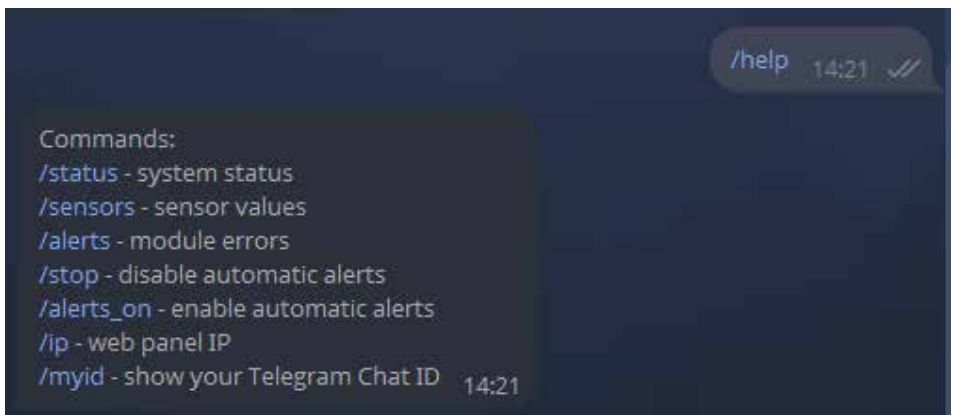

Рисунок 6. Демонстрація команди /help

Додаток В.

Демонстрація роботи Веб-панелі

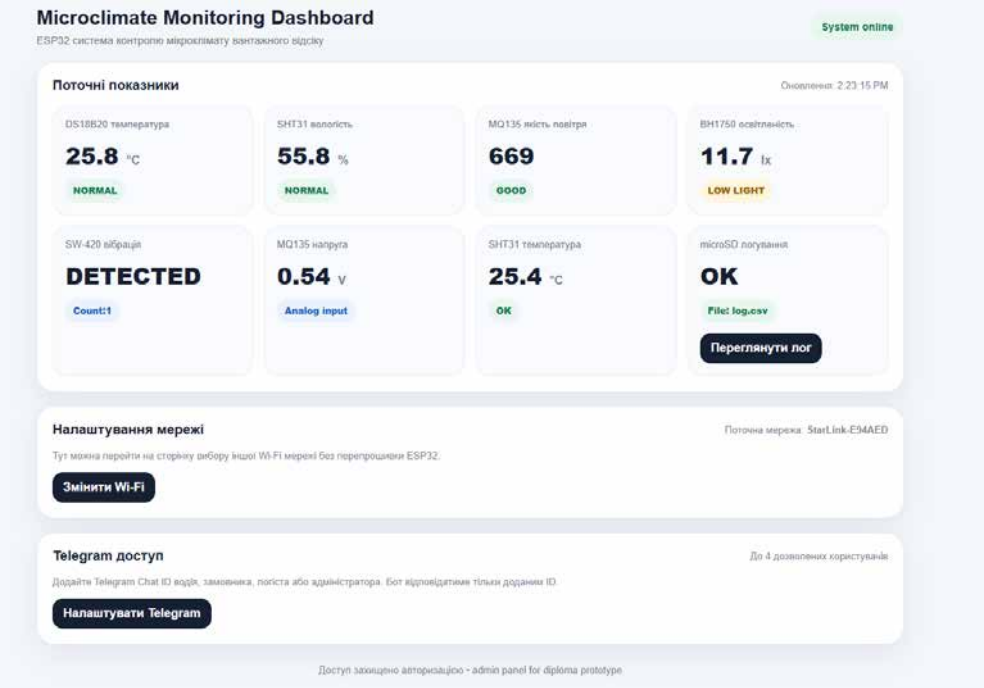

Рисунок 1. Демонсрація основної сторінки веб панелі

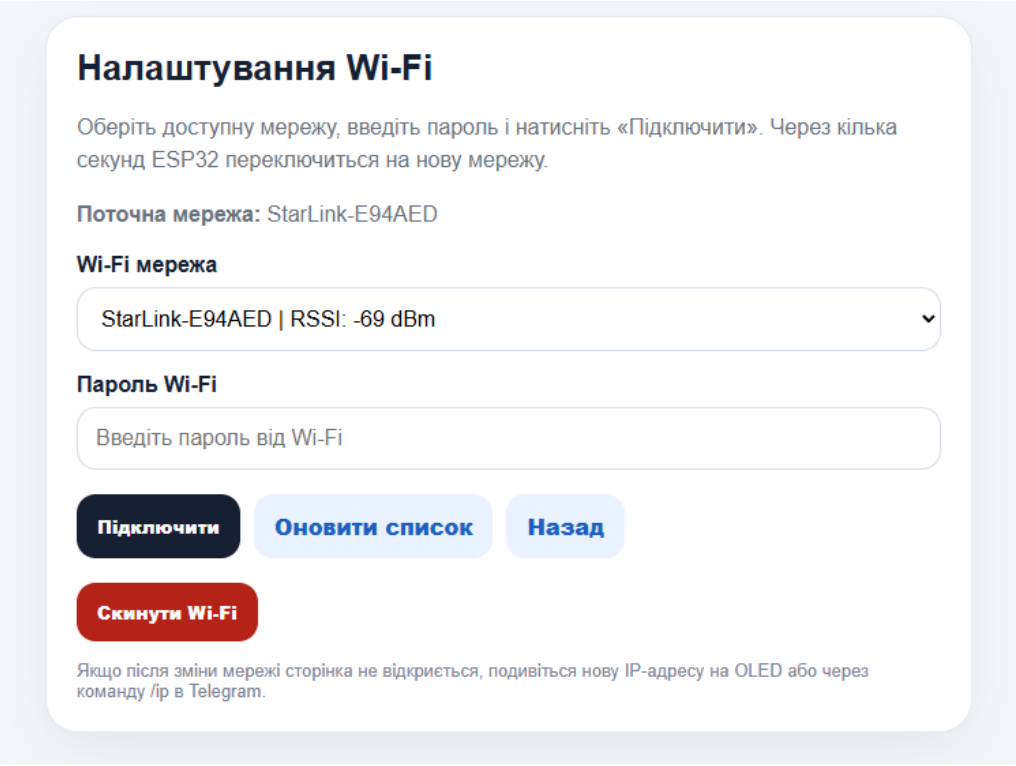

Рисунок 2. Демонстрація налаштуваннь WI-FI з’єднання

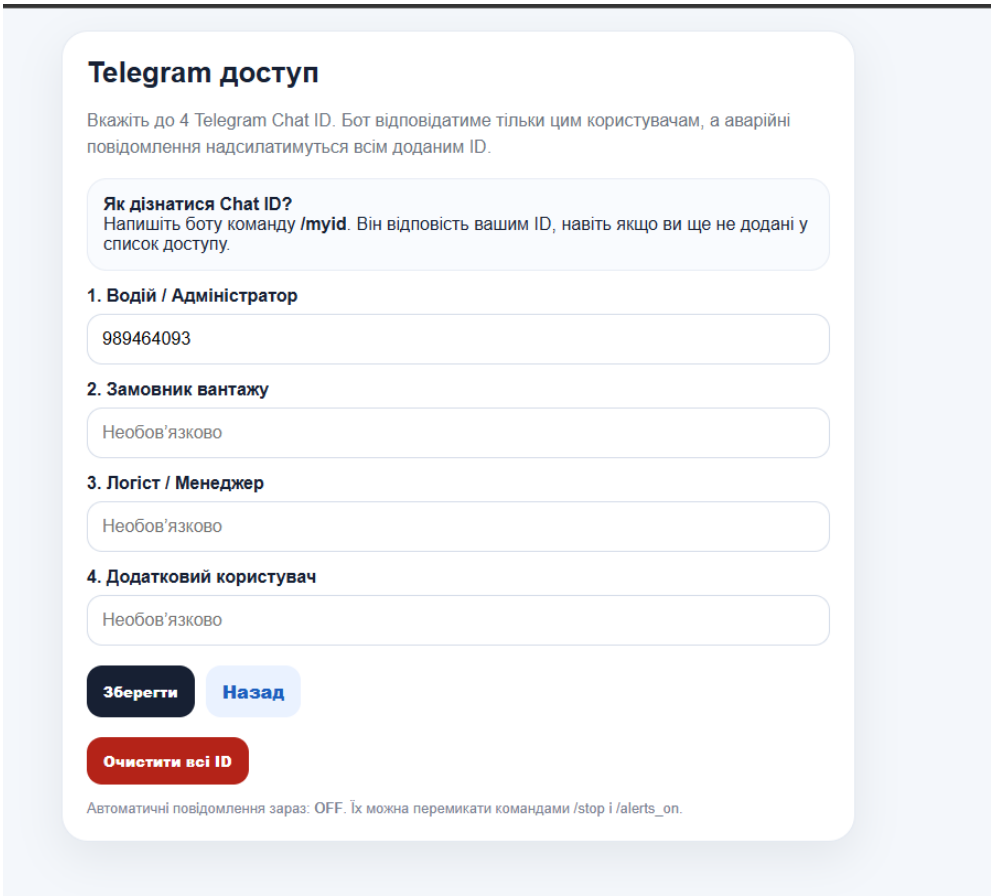

Рисунок 3. Демонстрація доступу до телеграм бота

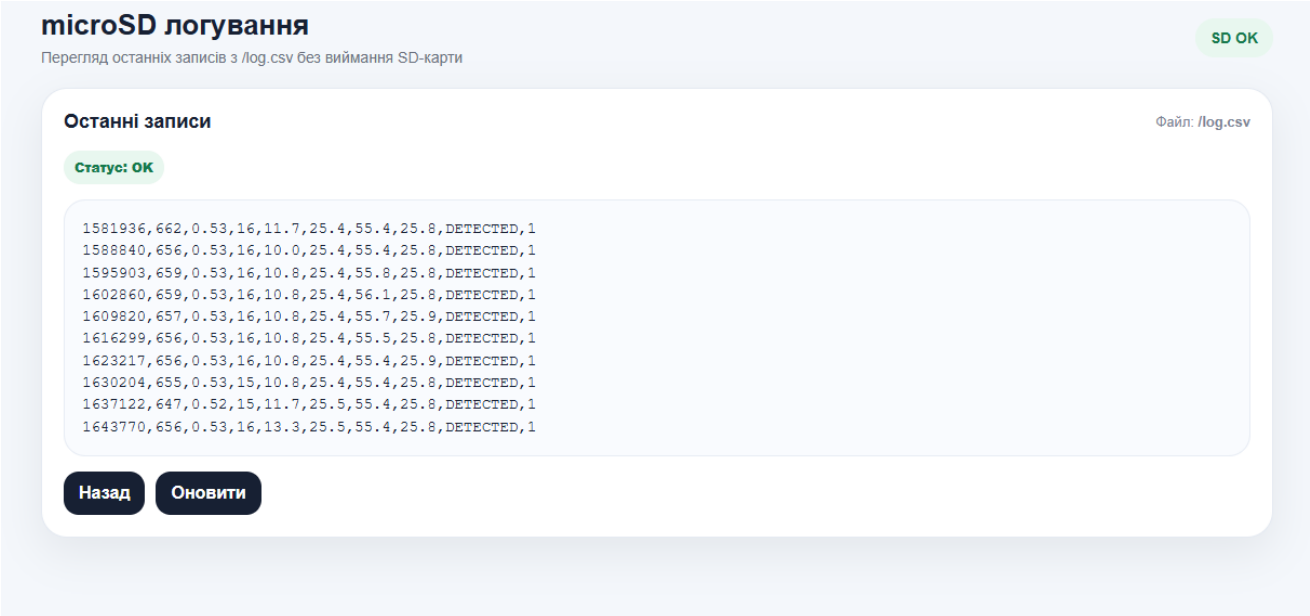

Рисунок 4. Демонстрація логування microSD