Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення телекомунікацій та електронних систем
(назва відділення)

Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітній ступінь)

на тему: **Розробка погодної станції на базі ESP32**

Виконав: студент VII курсу, групи КІб-703

Спеціальності **123 Комп'ютерна інженерія**
(шифр і назва спеціальності)

_____ Олександр РІЙ
(ім'я та прізвище)

Керівник Андрій НЕДОШИТКО
(ім'я та прізвище)

Рецензент _____
(ім'я та прізвище)

Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
імені ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерної інженерії
Освітній ступінь бакалавр
Освітньо-професійна програма: Комп'ютерна інженерія
Спеціальність: 123 Комп'ютерна інженерія
Галузь знань: 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"18" травня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рій Олкксандр Васильович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи **Розробка погодної станції на біза ESP32**

керівник роботи Недошитко Андрій Григорович
(прізвище, ім'я, по батькові)

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університетут імені Івана Пулюя» від 18.05.2026р №4/9-252.

2. Строк подання студентом роботи: 22 червня 2026 року.

3. Вихідні дані до роботи: завдання на проектування погодної станції на базі ESP32.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальна частина. Розробка технічного та робочого проєкту,, Спеціальний розділ. Економічний частина. Безпека життєдіяльності, Основи охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Струтцрна електрична схема, Схема електрична функціональна, Блок-схема алгоритму роботи системи, Таблиця техніко-економічних показників, Схема мережевої архітектури Telegran, Модель системи погодної станції (плакат).

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Ірина ЛЕЩИК методист, викладач		
Безпека життєдіяльності, основи охорони праці	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	18.05	
2	Збір і узагальнення інформації	21.05	
3	Написання першого розділу	25.05	
4	Розробка технічного та робочого проекту	1.06	
5	Написання спеціального розділу	8.06	
6	Розрахунок економічної частини	11.06	
7	Написання розділу охорони праці	12.06	
8	Виконання графічної частини	14.06	
9	Оформлення проекту	18.06	
10	Погодження нормоконтролю	19.06	
11	Попередній захист роботи	22.06	
12	Захист кваліфікаційної роботи		

7. Дата видачі завдання: 18 травня 2026 року

Студент _____ Олександр РІЙ
(підпис) (ім'я та прізвище)

Керівник роботи _____ Андрій НЕДОШИТКО
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

У кваліфікаційній роботі розроблено автономну погодну станцію на базі ESP32 для моніторингу параметрів навколишнього середовища та дистанційного доступу до даних через Telegram-бот.

У першому розділі проведено аналіз існуючих погодних станцій і сформовано вимоги до системи.

У другому розділі розроблено апаратну та програмну частини погодної станції, а також реалізовано її експериментальний макет.

У третьому розділі розроблено інструкцію з експлуатації та методику перевірки працездатності пристрою.

У четвертому розділі виконано економічне обґрунтування розробки.

У п’ятому розділі розглянуто питання охорони праці та заходи безпеки під час виконання робіт.

У результаті створено погодну станцію, що забезпечує збір, збереження та передачу даних користувачу.

Обсяг пояснювальної записки становить 83 сторінок. Графічна частина містить структурну та функціональну схеми, алгоритм роботи системи, таблицю техніко-економічних показників, схему мережевої архітектури Telegram і модель системи погодної станції(плакат) виконані на аркушах формату А1.

ABSTRACT

The qualification work presents an autonomous weather station based on the ESP32 microcontroller for environmental monitoring and remote access through a Telegram bot.

The first chapter analyzes existing weather stations and defines the requirements for the developed system.

The second chapter describes the hardware and software development of the weather station and its practical implementation.

The third chapter presents the operating manual and testing procedure for the device.

The fourth chapter provides the economic justification of the project.

The fifth chapter addresses occupational safety issues and safety measures during development activities.

As a result, a weather station capable of collecting, storing, and transmitting environmental data has been developed.

The explanatory note contains 83 pages. The graphical part consists of a structural diagram, a functional diagram, a system operation algorithm, a table of technical and economic indicators, a Telegram network architecture diagram, and a weather station system model, all presented on separate A1-size sheets.

ЗМІСТ

ВСТУП

Мікропроцесори є основою функціонування більшості сучасних електронних пристроїв — від побутової техніки до складних автоматизованих систем. Вони забезпечують обробку даних, керування процесами та взаємодію з користувачем, фактично виступаючи центральним елементом будь-якої інтелектуальної системи.

З розвитком технологій все більшої актуальності набуває моніторинг параметрів навколишнього середовища. Контроль температури, вологості, атмосферного тиску, освітленості та якості повітря є важливим як у наукових дослідженнях, так і в повсякденному житті. Особливо це актуально в умовах змін клімату та зростання вимог до екологічного контролю.

Перші метеорологічні спостереження проводились за допомогою простих механічних приладів, однак сучасні технології дозволяють створювати компактні автоматизовані системи, здатні не лише вимірювати параметри середовища, але й зберігати, аналізувати та передавати дані на відстань у режимі реального часу.

На сьогодні існує широкий вибір метеостанцій — від простих побутових пристроїв до складних професійних систем. Проте доступність мікроконтролерів та цифрових датчиків дозволяє створювати власні рішення, адаптовані під конкретні задачі користувача.

У даній кваліфікаційні роботі розробляється погодна станція на базі мікроконтролера, яка забезпечує вимірювання основних параметрів навколишнього середовища, їх збереження на зовнішньому носії та передачу користувачу за допомогою мережі Інтернет. Система також реалізує функції автоматичних попереджень при виході параметрів за допустимі межі та можливість дистанційного доступу через мобільний пристрій

1 ЗАГАЛЬНА ЧАСТИНА

1.1 Обґрунтування актуальності теми кваліфікаційної роботи

Мета данної кваліфікаційної роботи, розробка погодної станції на базі ESP32 для вимірювання екологічних параметрів навколишнього середовища.

Спостереження за довкіллям являється дуже важливим аспектом життя в 21 столітті, саме це спостереження дозволяє визначити і скласти плани на день, а також запобігти небажаним наслідкам таких як отруєння від забрудненого повітря під час аварій на промислових підприємствах чи природні катаклізми індикатором яких може бути змінна магнітного поля.

Перевагою данного проекта є постійний моніторинг, запис і аналіз даних наволишнього середовища, зручна передача сповіщень через Telegram. Пристрій може працювати в двох режимах: зовнішній (моніьлотнг довкілля) і внутріщній (спостереження в приміщенні).

Погодна станція буде оснащена такими функціями:

- вимірювання температури повітря;
- вимірювання вологості повітря;
- оцінка якості повітря;
- вимірювання освітленості;
- моніторинг магнітного поля;
- збереження даних на карту пам’яті;
- передача даних через мережу Wi-Fi;
- порівняння локальних вимірювань із даними онлайн-метеосервісів та даними попереднього року.

Запропонована погодна станція може використовуватися не тільки як інструмент для забезпечення комфортного життя але і як компактний

доступний за вартістю дослідницький стенд для молодих майбутніх метеорологів.

1.2 Аналітичний огляд існуючих рішень

Поміж пристроїв моніторингу природніх параметрів можна виділити декілька наступних прикладів:

1.Метеостанція KKMoon weather station WEA-46.

Домашня метеостанція KKMoon WEA-46 Weather Station (див. рис1.1) призначена для контролю температури та вологості повітря у різних приміщеннях або на відкритому повітрі. Пристрій комплектується базовим блоком та кількома бездротовими зовнішніми датчиками, які можуть розміщуватися як поза будівлею, так і всередині різних приміщень, наприклад у гаражі чи на кухні. Виміряні параметри передаються до основного блоку, де відображаються окремо для кожного датчика. Зовнішні модулі також оснащені невеликими дисплеями для локального перегляду показників.

Рисунок 1.1 – Метеостанція WEA-46[2]

Характеристики метеостанції KKMoon WEA-46 Weather Station:

- Тип пристрою — електронна метеостанція
- Діапазон вимірювання температури у приміщенні — 0…50 °C

- Діапазон вимірювання зовнішньої температури — –40…70 °C

- Діапазон вимірювання відносної вологості у приміщенні — 20…95 %

- Діапазон вимірювання відносної вологості зовнішнього повітря — 20…95 %

- Живлення базової станції — 4,5 В

- Живлення зовнішнього датчика — 3 В

- Максимальна дальність передачі сигналу від датчика до базового блоку — до 20 м

Комплектація пристрою

- Базовий блок метеостанції — 1 шт

- Бездротові зовнішні датчики — 3 шт

- Базова станція — 106 г

- Зовнішній датчик — 40 г

- Вартість пристрою: 379 грн[3].

2.Метеостанція Techno Line WS9138 .

Метеостанція La Crosse WS9138 Weather Station (див. рис. 1.2) є компактним електронним пристроєм для контролю основних параметрів навколишнього середовища. Вона оснащена цифровим дисплеєм, на якому відображаються дані про температуру повітря в приміщенні та на вулиці, відносну вологість, прогноз погоди та поточний час. Передача інформації про зовнішню температуру здійснюється бездротовим способом від зовнішнього датчика до базової станції. Живлення пристрою забезпечується батарейками типу AAA. Пристрій характеризується простотою встановлення та експлуатації, а також не потребує постійного підключення до мережі живлення. Отримані дані відображаються в режимі реального часу, що забезпечує зручний контроль погодних умов користувачем.

Рисунок 1.2 – Метеостанція La Crosse WS9138 Weather Station[4]

Характеристики метеостанції Techno Line WS9138:

- Діапазон вимірювання температури у приміщенні — -9,9...50 °C
- Діапазон вимірювання зовнішньої температури — -40...70 °C
- Діапазон вимірювання відносної вологості у приміщенні — 20…95 %
- Діапазон вимірювання відносної вологості зовнішнього повітря — 20…95 %
- Живлення базової станції — 3 В
- Живлення зовнішнього датчика — 3 В
- Максимальна дальність передачі сигналу від датчика до базового блоку — до 50 м
- Розмір
- Базова станція — 90 x 93 x 45 мм
- Зовнішній датчик — 40 x 90 x 23 мм

  Маса пристрою
- Базова станція — 85 г
- Зовнішній датчик — 30 г
- Вартість пристрою: 2 180 грн[5].

2. РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЕКТУ

2.1 Аналіз технічного завдання КРБ

Основною ідеєю даного пристрою є розробка компактної погодної станції, яка здійснює вимірювання параметрів навколишнього середовища, їх збереження та передачу користувачу через бездротову мережу.

Пристрій виконаний у вигляді автономного блоку, що розміщується в корпусі з можливістю встановлення як у приміщенні, так і на відкритому повітрі. Конструкція корпусу передбачає наявність отворів для доступу повітря до датчиків, а також елементів для фіксації електронних компонентів.

Опис конструкції пристрою

Конструкція погодної станції виконана у вигляді корпусу, всередині якого розміщені основні електронні модулі та датчики. Розташування елементів у корпусі показано на рисунку 2.1.

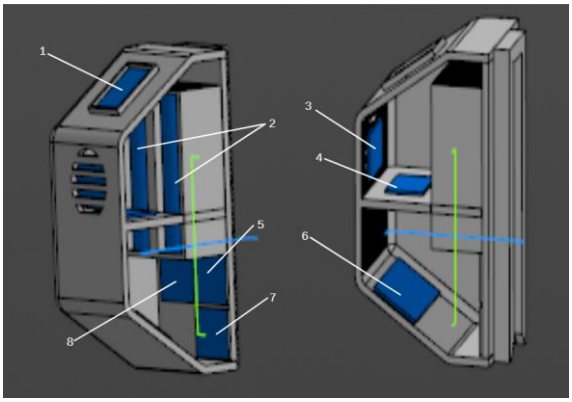


Рисунок 2.1 – Корпус погодної станції

До складу пристрою входять такі елементи:

1. Датчик освітленості використовується для вимірювання освітленості в люксах;
2. Акумулятори для автономного живлення станції;
3. Датчик температури, вологості та якості повітря;

4. Магнітометр для визначення магнітного коливання по осях X\У\Z;
5. Модуль зарядки акумуляторів;
6. Мікроконтролер ESP32, який забезпечує обробку даних, керування системою та бездротовий зв’язок;
7. Модуль карти пам’яті MicroSD для збереження виміряних даних;
8. Підвищувальний DC-DC модуль для стабілізації напруги.

Структура функціональних вузлів погодної станції зображена на рисунку 2.2.

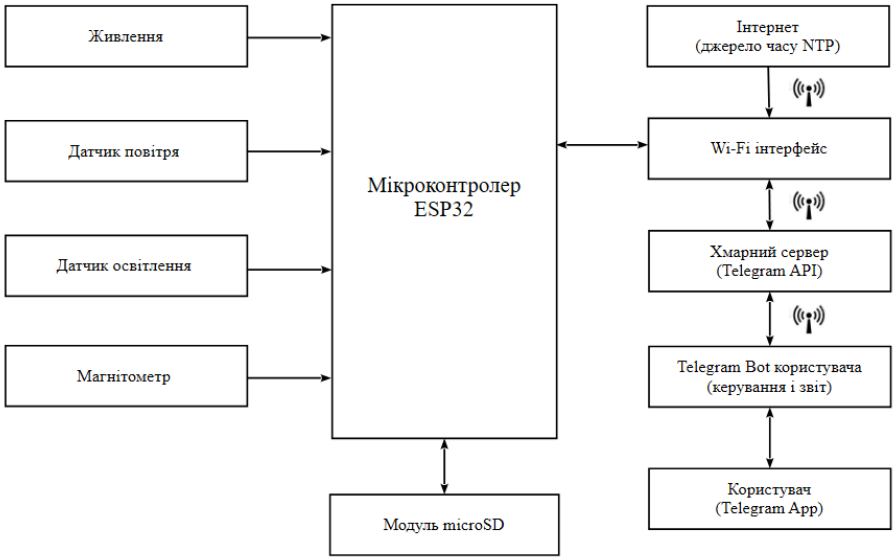


Рисунок 2.2 – Структурна схема погодної станції

Взаємодія з користувачем

Взаємодія користувача з погодною станцією здійснюється за допомогою мобільного пристрою через мережу Інтернет. Це дозволяє:

- отримувати поточні значення вимірюваних параметрів;
- переглядати збережені дані;
- змінювати налаштування роботи пристрою;
- отримувати сповіщення у разі зміни стану середовища.

Особливості системи

Передача даних здійснюється через бездротову мережу з використанням вбудованого Wi-Fi модуля, що дозволяє отримувати інформацію з будь-якої точки за наявності доступу до Інтернету.

2.2 Опис і обґрунтування вибору елементної бази

В якості основного керуючого елемента погодної станції використовується мікроконтролер ESP32. Даний мікроконтролер є сучасним високопродуктивним рішенням, яке широко застосовується в системах Інтернету речей (IoT).

ESP32 побудований на базі 32-розрядної архітектури та має двоядерний процесор, що дозволяє ефективно виконувати декілька задач одночасно. Це є важливим для даного проєкту, оскільки пристрій повинен одночасно здійснювати зчитування даних з датчиків, обробку інформації, запис на карту пам’яті та передачу даних через мережу Wi-Fi.

Однією з ключових особливостей ESP32 є наявність вбудованого модуля Wi-Fi, що дозволяє реалізувати бездротову передачу даних без використання додаткових зовнішніх модулів. Це значно спрощує апаратну частину пристрою та зменшує його вартість.

Мікроконтролер також підтримує велику кількість інтерфейсів, таких як I2C, SPI, UART, ADC та інші, що забезпечує зручне підключення різноманітних сенсорів і периферійних пристроїв.

Основні характеристики:

- тактова частота до 240 МГц;
- двоядерний процесор;
- вбудований Wi-Fi (802.11 b/g/n);
- підтримка Bluetooth;
- кількість GPIO — до 34;

- наявність інтерфейсів I2C, SPI, UART[6];
- низьке енергоспоживання.

Обґрунтування вибору:

Мікроконтролер ESP32 обрано завдяки наявності вбудованого модуля Wi-Fi, що дозволяє реалізувати передачу даних без використання додаткових зовнішніх пристроїв. Важливою перевагою є також достатній обсяг оперативної та флеш-пам’яті, що забезпечує можливість обробки даних, роботи з мережею та реалізації додаткового функціоналу, зокрема взаємодії з Telegram-ботом і збереження конфігурацій.

Крім того, ESP32 характеризується високою швидкодією та продуктивністю завдяки двоядерній архітектурі та високій тактовій частоті, що дозволяє ефективно виконувати декілька задач одночасно без затримок у роботі системи.

Використання даного мікроконтролера забезпечує реалізацію всіх необхідних функцій погодної станції в межах одного пристрою, що спрощує апаратну частину та підвищує надійність системи.

Датчик повітря

Для моніторингу параметрів повітряного середовища в погодній станції було взято датчик BME680. Він вимірює температуру повітря, відносну вологість та атмосферний тиск, а також оцінює якість повітря за вмістом летких органічних сполук (VOC). На рисунку 2.3 зображено зовнішній вигляд датчика BME680.

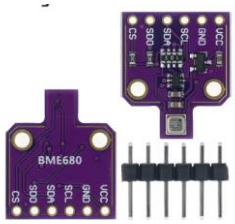


Рисунок 2.3 – Модуль BME680[7].

Характеристики:

- вимірювання температури повітря: –40…+85 °C;
- вимірювання відносної вологості повітря: 0…100 %;
- вимірювання атмосферного тиску: 300…1100 гПа;
- оцінка якості повітря (VOC);
- інтерфейс: I2C;
- напруга живлення: 3.3 В[5].

Позначення модуля BME680. На схемі зображене на рисунку 2.4.

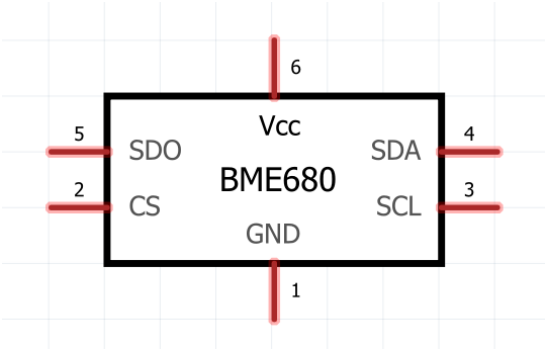


Рисунок 2.4 – Зображення модуля BME680 на схемі.

Датчик освітлення

В якості датчика освітлення було вибрано BH1750 який призначений для вимірювання рівня освітленості навколишнього середовища в люксах. Отримані дані дозволяють оцінювати зміну світлових умов протягом доби та вплив зовнішніх факторів, таких як хмарність або штучне освітлення. На рисунку 2.5 зображений зовнішній вигляд датчика освітлення BH1750.

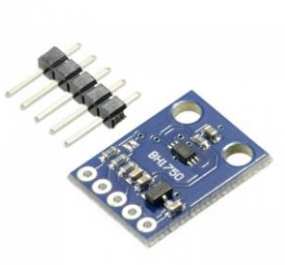


Рисунок 2.5 – Датчик освітлення BH1750[8]

Основні характеристики:

- діапазон вимірювання: 1…65535 лк;
- інтерфейс: I2C;
- напруга живлення: 3.3–5 В;
- висока точність вимірювання освітленості[6].

Датчик підключається до мікроконтролера ESP32 через інтерфейс I2C, що дозволяє використовувати спільну шину даних разом з іншими сенсорами (див. рис. 2.6).

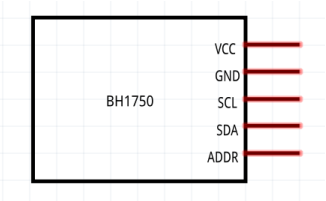


Рисунок 2.6 – Зображення модуля BH1750 на схемі.

Магнітометр

В якості магнітометра (див. рис. 2.7) обрано датчик HMC5883L, який вимірює складові магнітного поля по трьох осях (X, Y, Z). Отримані дані використовуються для визначення орієнтації пристрою, фіксації аномальних змін та моніторингу техногенних чи природних впливів на ранніх етапах.

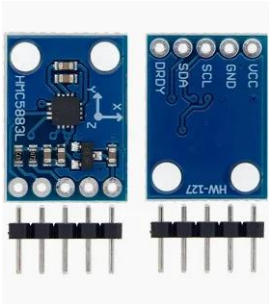


Рисунок 2.7 – Магнітометр HMC5883L[9].

Основні характеристики:

- тип датчика: тривісний магнітометр;

- діапазон вимірювання: ±8 Гс;
- роздільна здатність: до 5 мГс;
- інтерфейс передачі даних: I2C;
- напруга живлення: 3.3–5 В;
- низьке енергоспоживання;
- висока чутливість до змін магнітного поля[7].

На рисунку 2.8 зображено позначення HMC5883 на схемі.

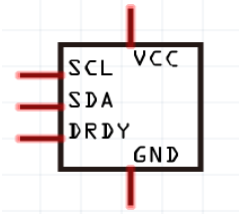


Рисунок 2.8 – Позначення HMC5883 на схемі

Модуль MicroSD карти пам’яті

Модуль MicroSD використовується в погодній станції для збереження результатів вимірювань у вигляді лог-файлів. Це дозволяє накопичувати історичні дані та виконувати подальший аналіз змін параметрів навколишнього середовища.

Запис даних здійснюється у структурованому форматі з фіксацією часу вимірювання та значень усіх сенсорів. Це забезпечує можливість відстеження динаміки змін погодних умов протягом тривалого часу.

Модуль (див. рис. 2.9) підключається до мікроконтролера ESP32 через інтерфейс SPI, що забезпечує швидку передачу даних та стабільну роботу з файловою системою.

Рисунок 2.9 – Модуль microSD[10]

Характеристики:

- підтримка карт пам’яті MicroSD;
- файлові системи: FAT16 / FAT32;
- інтерфейс: SPI;
- напруга живлення: 3.3–5 В;
- швидка передача даних;
- можливість тривалого збереження інформації[8].

На рисунку 2.10 зображенно позначення MicroSD на схемі.

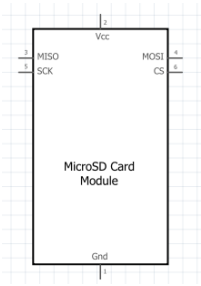


Рисунок 2.10 – Позначення MicroSD на схемі

Модуль зарядки TP4056

Модуль TP4056 використовується для зарядки літій-іонного акумулятора, який забезпечує автономну роботу погодної станції. Він реалізує контроль процесу зарядки та захист акумулятора від перезаряду і перерозряду.

Модуль зарядки (див. рис. 2.11-2.12) дозволяє заряджати акумулятор від джерела живлення 5 В (наприклад, USB), що робить систему зручною у використанні та універсальною.

Рисунок 2.11 – Модуль зарядки TP4056[11]

Основні характеристики:

- тип акумулятора: Li-ion / Li-Po;
- напруга живлення: 5 В;
- максимальний струм зарядки: до 1 А;
- контроль заряду акумулятора;
- захист від перезаряду та перерозряду;
- компактні розміри[9].

-

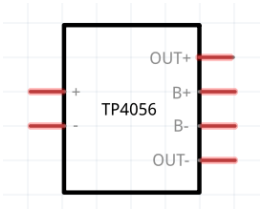


Рисунок 2.12 – Позначення TP4056 на схемі

Підвищуючий перетворювач MT3608 (див. рис. 2.13).

MT3608 використовується для підвищення напруги з акумулятора до стабільного значення, необхідного для живлення мікроконтролера. Модуль дозволяє перетворювати напругу 3.0–4.2 В у стабільні 5 В.

Регулювання вихідної напруги здійснюється за допомогою підстроювального резистора, що дозволяє точно налаштувати необхідний рівень живлення. Модуль забезпечує стабільну роботу пристрою навіть при зміні напруги акумулятора.

Рисунок 2.13 – перетворювач MT3608[12].

Основні характеристики:

- тип: DC-DC підвищуючий перетворювач (див. рис. 2.14);
- вхідна напруга: 2–24 В;
- вихідна напруга: до 28 В (регульована);
- максимальний вихідний струм: до 2 А;
- коефіцієнт корисної дії: до 90%;
- компактні розміри[10].

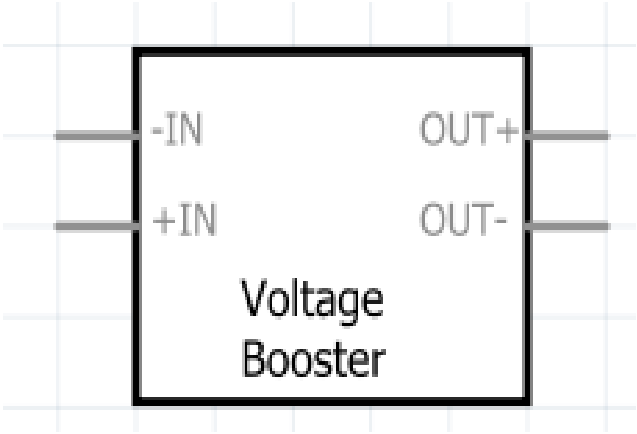


Рисунок 2.14 – позначення перетворювача MT3608 на схемі

2.3 Розробка функціональної схеми пристрою

Функціональна схема є одним із основних проектних документів, що визначає загальну структуру розробленої системи та взаємозв’язок між її основними компонентами.

Вона використовується для графічного представлення принципу роботи пристрою та передачі інформації між електронними модулями, які входять до складу системи.

Функціональна схема метеорологічної станції наведена на рисунку 2.15.

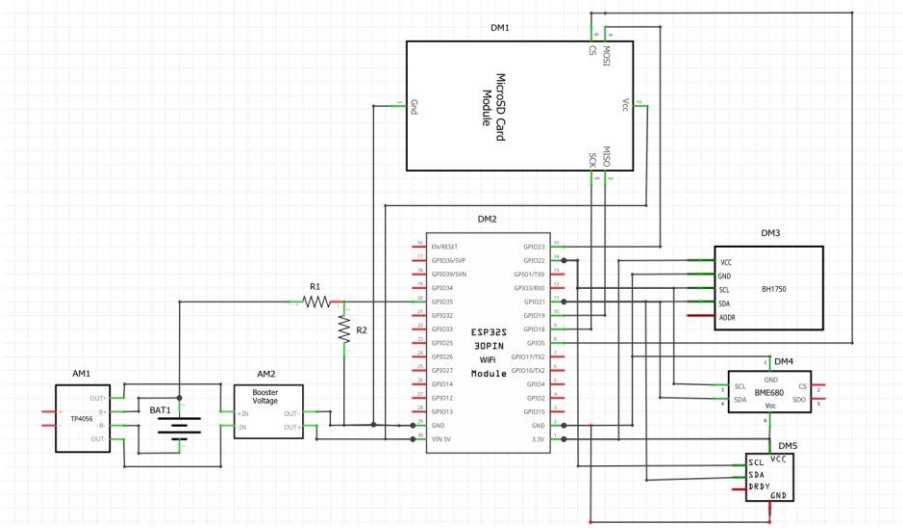


Рисунок 2.15 – Функціональна схема пристрою

DM1 – модуль карти пам’яті MicroSD, призначений для збереження журналу вимірювань та конфігураційних параметрів системи.

DM2 – контролер ESP32, який є основним керуючим елементом системи. Контролер здійснює обробку даних із датчиків, керування периферійними модулями, збереження інформації та передачу інформації через мережу WiFi.

DM3 – модуль датчика освітленості BH1750, використовується для вимірювання рівня освітлення навколишнього середовища в люксах.

DM4 – модуль датчика BME680, призначений для вимірювання температури, вологості, атмосферного тиску та оцінки якості повітря за допомогою газового сенсора.

DM5 – модуль магнітометра HMC5883L, призначений для вимірювання магнітного поля по трьох осях X, Y та Z.

AM1 – модуль зарядки TP4056, використовується для заряджання акумуляторної батареї та забезпечення стабільного живлення пристрою.

AM2 – підвищувальний DC-DC перетворювач MT3608, призначений для стабілізації та підвищення напруги живлення до необхідного рівня для коректної роботи системи.

BAT1 – акумуляторна батарея типу Li-Ion або Li-Po, призначена для автономного живлення метеостанції.

R1, R2 – резистивний дільник напруги (100kOm і 100kOm), призначений для зниження напруги акумуляторної батареї до безпечного рівня для подальшого вимірювання контролером ESP32 через аналоговий вхід GPIO35.

Розроблена функціональна схема забезпечує взаємодію всіх модулів системи, збір та обробку метеорологічних даних, їх збереження на карті пам’яті та передачу користувачу за допомогою бездротового мережевого з’єднання. Таке інженерне рішення гарантує високу стабільність обміну даними між мікроконтролером та периферією.

2.3 Розробка Telegram-бота для управління станцією

Telegram-бот використовується як основний інтерфейс взаємодії користувача з метеостанцією. Даний спосіб управління дозволяє отримувати інформацію про стан навколишнього середовища, змінювати параметри роботи пристрою та отримувати аварійні повідомлення через мережу Інтернет без необхідності створення окремого мобільного застосунку.

Для реалізації системи було використано месенджер Telegram та офіційний сервіс створення ботів BotFather (див. рис. 2.16). Створення бота виконується шляхом реєстрації нового бота через команду `/newbot`, після чого користувач отримує унікальний токен доступу до Telegram API. Даний токен використовується контролером ESP32 для обміну повідомленнями із сервером Telegram.

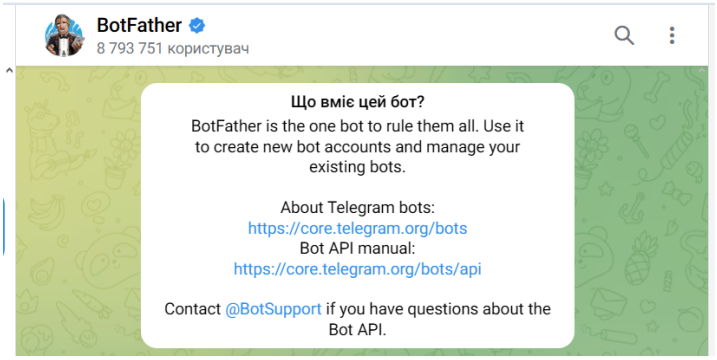


Рисунок 2.16 – Офіційний сервіс створення ботів BotFather

У програмному коді метеостанції для роботи з Telegram API використовується бібліотека UniversalTelegramBot, яка забезпечує:

- отримання повідомлень від користувача;
- обробку команд;
- передачу текстових повідомлень;
- роботу з інтерактивними кнопками (Inline Keyboard).

Після підключення до мережі WiFi контролер ESP32 періодично виконує перевірку нових повідомлень користувача та обробляє отримані команди. Основні команди Telegram-бота наведені нижче (див. рис. 2.17):

- `/find` – пошук записів у журналі вимірювань на карті пам’яті SD;
- `/daily` – встановлення часу щоденного звіту;
- `/mode` – вибір режиму роботи метеостанції;
- `/alerts` – увімкнення або вимкнення аварійних повідомлень;
- `/calibrate` – запуск процедури калібрування датчиків.

Рисунок 2.17 – Вигляд Telegram-чату з ботом

Для спрощення взаємодії користувача з системою були реалізовані інтерактивні кнопки Telegram. Кнопки дозволяють швидко отримувати інформацію про температуру, вологість, рівень освітлення, якість повітря та показники магнітометра без необхідності введення текстових команд вручну (див. рис. 2.18).

Рисунок 2.18 – Інтерактивне меню Telegram-бота

Формування повідомлень виконується безпосередньо контролером ESP32. Дані із датчиків обробляються програмою, після чого передаються користувачу у вигляді текстового повідомлення через Telegram API.

2.4 Розробка алгоритму системи

Для забезпечення безперервного моніторингу параметрів навколишнього середовища було розроблено алгоритм роботи погодної станції, який забезпечує збір даних із датчиків, їх обробку, збереження на карту пам'яті та передачу користувачу через мережу Інтернет. Алгоритм побудований таким чином, щоб система могла працювати автономно протягом тривалого часу без втручання користувача.

Основою системи є мікроконтролер ESP32, який координує роботу всіх підключених модулів та виконує обробку отриманої інформації. Після подачі живлення контролер виконує початкову ініціалізацію обладнання та

програмних модулів, після чого переходить у режим безперервного циклічного виконання основної програми.

На етапі запуску системи відбувається перевірка працездатності підключених модулів, зчитування конфігураційних параметрів з карти пам'яті та встановлення мережевого з'єднання через WiFi. Після успішного підключення до мережі здійснюється синхронізація внутрішнього часу контролера із сервером точного часу NTP, що дозволяє прив'язувати всі записи до реальної дати та часу.

Алгоритм ініціалізації системи складається з таких етапів:

1. Завантаження необхідних програмних бібліотек.
2. Ініціалізація інтерфейсів введення-виведення.
3. Ініціалізація карти пам'яті MicroSD.
4. Ініціалізація датчика BME680.
5. Ініціалізація датчика BH1750.
6. Ініціалізація магнітометра HMC5883L.
7. Завантаження параметрів конфігурації.
8. Підключення до бездротової мережі WiFi.
9. Синхронізація часу через NTP-сервер.
10. Перехід до основного циклу роботи.

Після завершення процедури ініціалізації система переходить до циклічного опитування датчиків. У процесі роботи контролер отримує значення температури повітря, відносної вологості, атмосферного тиску, освітленості, магнітного поля та показники якості повітря. Додатково виконується контроль рівня заряду акумуляторної батареї за допомогою аналогового входу контролера та резистивного дільника напруги.

Зчитані дані тимчасово зберігаються у внутрішній структурі даних, що дозволяє використовувати їх для формування звітів, перевірки аварійних ситуацій та запису до журналу вимірювань.

Для забезпечення накопичення статистичної інформації реалізовано механізм автоматичного логування даних. Запис показників здійснюється щогодини, при цьому для кожної календарної дати створюється окремий файл журналу. Такий підхід спрощує пошук інформації та дозволяє зменшити розмір окремих файлів.

Алгоритм збереження даних виконується у такій послідовності:

1. Отримання поточної дати та часу.
2. Зчитування показників усіх датчиків.
3. Формування рядка журналу вимірювань.
4. Відкриття файлу поточної дати.
5. Запис даних на карту пам'яті.
6. Закриття файлу.
7. Повернення до основного циклу роботи.

Однією з головних функцій системи є можливість дистанційного доступу до інформації через Telegram-бот. Використання Telegram дозволяє користувачу взаємодіяти з погодною станцією в будь-якій точці світу, де є доступ до мережі Інтернет, без необхідності розробки окремого мобільного застосунку.

У процесі роботи контролер періодично перевіряє наявність нових повідомлень від користувача. При отриманні команди виконується її аналіз та відповідна обробка. Система підтримує перегляд поточних показників датчиків, пошук записів у журналі вимірювань, зміну режимів роботи, налаштування часу щоденного звіту та запуск процедури калібрування датчиків.

Алгоритм роботи Telegram-бота включає:

1. Перевірку наявності нових повідомлень.
2. Аналіз отриманої команди.
3. Визначення необхідної дії.
4. Виконання відповідної функції.

5. Формування тексту відповіді.

6. Відправлення повідомлення користувачу.

Для підвищення інформативності системи реалізовано функцію автоматичного формування щоденного звіту. У визначений користувачем час контролер формує повідомлення, яке містить актуальні значення всіх контрольованих параметрів, після чого відправляє його через Telegram-бот.

Алгоритм формування щоденного звіту складається з таких етапів:

1. Отримання поточного часу.

2. Порівняння поточного часу із встановленим часом звіту.

3. Зчитування показників датчиків.

4. Формування текстового звіту.

5. Передача звіту користувачу через Telegram.

Для оперативного реагування на небезпечні або небажані зміни параметрів навколишнього середовища реалізовано систему тригерних повідомлень. Система автоматично контролює показники датчиків та порівнює їх із встановленими пороговими значеннями.

У випадку перевищення або зниження контрольованих параметрів нижче допустимого рівня користувач отримує відповідне попередження. Для уникнення багаторазового дублювання повідомлень використовуються спеціальні прапорці стану, які блокують повторне відправлення повідомлення до моменту повернення параметра в нормальний діапазон.

Система підтримує два режими роботи — внутрішній та зовнішній. Залежно від обраного режиму використовуються різні порогові значення для формування попереджень, що дозволяє адаптувати роботу пристрою до різних умов експлуатації.

До контрольованих подій належать:

- погіршення якості повітря;
- підвищена або знижена вологість;
- недостатній або надмірний рівень освітлення;

- можливість випадання опадів;
- критично низький заряд акумуляторної батареї.

Таким чином розроблений алгоритм забезпечує автоматичний збір, обробку, збереження та передачу інформації користувачу в режимі реального часу. Використання мережевих технологій та автономного живлення дозволяє експлуатувати систему як у приміщенні, так і на відкритій місцевості, забезпечуючи постійний контроль параметрів навколишнього середовища. Окрім цього, логіка програми передбачає перехід мікроконтролера в режими зниженого енергоспоживання між циклами опитування датчиків, що оптимізує витрату заряду акумулятора.

Програмно реалізовано алгоритми перевірки готовності системи, які виключають ризик втрати чи пошкодження даних під час критичного падіння напруги. Загальний алгоритм роботи метеостанції наведено на листі графічної частини «Блок-схема алгоритму роботи системи».

2.5 Написання текстів програми

Програмне забезпечення погодної станції розроблено мовою C++ у середовищі Arduino IDE. Для забезпечення роботи мережевих функцій, датчиків та накопичувача даних використовуються спеціалізовані бібліотеки.

```
#include <WiFi.h> //для підключення до бездротової мережі Wi-Fi
#include "time.h" //для захищеного HTTPS-з’єднання
#include <SPI.h> //для швидкох роботи з периферсією (з microSD)
#include <SD.h> //для роботи з microSD
#include <WiFiClientSecure.h> //для захищеного HTTPS-з’єднання
#include <UniversalTelegramBot.h> //для роботи з Telegram-Bot
#include <Wire.h> //для роботи з датчиками через шину I2C
#include <Adafruit_Sensor.h> //для переведення сирих даних
#include <Adafruit_BME680.h> //для роботи з датчиком якості повітря
#include <Adafruit_HMC5883_U.h> // для роботи з магнітометром
#include <BH1750.h> // для роботи з люксметром
```

Для зберігання параметрів підключення та конфігураційних даних використовуються глобальні змінні.

```
String ssid = ""; //ім'я мережі
String password = ""; //пароль мережі
String BOT_TOKEN = ""; //токен чат бота
String CHAT_ID = ""; //ID чата
//сервер синхронізації точного часу
const char* ntpServer = "pool.ntp.org";
WiFiClientSecure client; //клієнт для HTTPS-з'єднання з Telegram API
UniversalTelegramBot* bot; об'єкт Telegram-бота для обробки команд
Adafruit_BME680 air; //об'єкт датчика якості повітря
BH1750 light; //об'єкт люксометра
//об'єкт магнітометра
Adafruit_HMC5883_Unified magnet = Adafruit_HMC5883_Unified(12345);
bool gasAlertSent = false; //флаг, що запобігає повторному надсиланню алерту газу
bool humAlertSent = false; //флаг алерту вологості
bool luxAlertSent = false; //флаг алерту освітленості
bool rainAlertSent = false; //флаг алерту можливих опадів
bool batAlertSent = false; //флаг алерту низького заряду батареї
bool alertsEnabled = true; //вмикання/вимикання системи алертів
float gasNorm = 0; //калібрувальне значення нормального рівня газу
//зміщення магнітометра по осях X/Y/Z
float offsetX = 0, offsetY = 0, offsetZ = 0;
float batCalibr = 1.0; //коефіцієнт калібрування вимірювання батареї
int lastLoggedHour = -1; //запам'ятовування останньої години запису
int lastDailyDay = -1; //день останнього щоденного звіту
int dailyHour = 7; //година автоматичного щоденного звіту
int dailyMinute = 0; //хвилина автоматичного щоденного звіту
unsigned long alertInterval = 60000; //інтервал перевірки алертів
unsigned long lastAlertTime = 0; //час останньої перевірки алертів
unsigned long lastTelegramCheck = 0; //час останньої перевірки Telegram
const int SD_CS = 5; //пін керування microSD (Chip Select)
const int BAT_PIN = 35; //аналоговий пін вимірювання напруги батареї
```

Для організації даних, отриманих із датчиків, використовується структура SensorData. Вона об'єднує всі вимірювані параметри в єдиний об'єкт, що спрощує передачу та обробку даних у програмі.

```
struct SensorData {
  float temp; //температура повітря
  float hum;  //відносна вологість
  float gas;  //рівень газів (якість повітря)
  float pres; //атмосферний тиск
  float lux;  //освітленість
  float mX;   //магнітне поле по осі X
  float mY;   //магнітне поле по осі Y
  float mZ;   //магнітне поле по осі Z
  float bat;  //рівень заряду батареї
};
```

Для визначення режиму роботи пристрою (внутрішній або зовнішній) використовується перелік (enum):

```
enum Mode {
  INDOOR,  //режим роботи в приміщенні
  OUTDOOR  //режим роботи на відкритому повітрі
      };
```

Поточний режим роботи пристрою зберігається у змінній deviceMode, яка за замовчуванням встановлена в режим INDOOR.

```
Mode deviceMode = INDOOR; //по замовчуванню
```

Для організації структури програми використовуються прототипи функцій, які оголошуються на початку коду. Це дозволяє викликати функції до їх фактичного опису у програмі та покращує читабельність коду.

```
SensorData readSensors();
//зчитування даних з усіх датчиків
String buildData(SensorData s, String type = "log");
//формування текстового представлення даних
String findSD(String query); //пошук даних на SD-карті
void saveConfig(String key, String newValue);
//збереження конфігураційних параметрів

void saveToSD(String data); //запис даних у файл на SD-карті
void checkAlerts(SensorData s); //перевірка умов спрацювання алертів
void calibrate(); //калібрування датчиків
float batteryCharge(); //розрахунок рівня заряду батареї
```

Функція setup() виконує початкову ініціалізацію всіх апаратних та програмних компонентів системи перед основним циклом роботи.

```
void setup() {
 Serial.begin(115200);//ініціалізація послідовного порту
 Wire.begin(21, 22); //ініціалізація I2C шини
```

Далі виконується ініціалізація SD-карти та перевірка наявності файлової системи для зберігання логів:

```
  if (!SD.begin(SD_CS)) {
  Serial.println("SD init failed"); //повідомлення про помилку
 }
 if (!SD.exists("/logs")) { //перевірка існування директорії логів
   SD.mkdir("/logs"); //створення директорії, якщо вона відсутня
 }
```

Після цього ініціалізуються датчики навколишнього середовища:

```
  if (!air.begin(0x77)) {
   Serial.println("BME680 init failed"); //помилка  датчика BME680
 }
 else {
  air.setGasHeater(320, 150); //налаштування газового сенсора
 }
 if (!magnet.begin()) {
 Serial.println("HMC5883 init fail"); //помилка магнітометра
 }
 if (!light.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x23)) {
   Serial.println("BH1750 init fail"); //помилка ініціалізації датчика освітленості
 }
```

Потім завантажується конфігурація системи та ініціалізується Telegram-бот:

```
 loadConfig(); //завантаження конфігураційних параметрів з SD-карти
 client.setInsecure(); //відключення перевірки SSL сертифікатів
```

```
  bot = new UniversalTelegramBot(BOT_TOKEN, client); //створення об’єкта Telegram-бота
```

Виконується підключення до Wi-Fi мережі:

```
   WiFi.begin(ssid.c_str(), password.c_str()); //ініціалізація підключення до Wi-Fi
  unsigned long wifiStart = millis(); //фіксація часу початку підключення
  while (WiFi.status() != WL_CONNECTED &&
    millis() - wifiStart < 20000) { //очікування підключення до мережі
    delay(500); //затримка між спробами підключення
    Serial.print("."); //індикація процесу підключення
  }
```

Після успішного підключення налаштовується синхронізація часу та часовий пояс:

```
  configTime(0, 0, ntpServer); //синхронізація часу через NTP сервер
  setenv("TZ", "EET-2EEST,M3.5.0/3,M10.5.0/4", 1); //встановлення часового поясу
  tzset(); //застосування налаштувань часового поясу
```

Завершальним етапом є налаштування ADC для коректного вимірювання напруги батареї:

```
  analogSetAttenuation(ADC_11db); //розширення діапазону вимірювання
```

Початок циклу: фіксується поточний час роботи мікроконтролера та оголошується структура для роботи з системним часом.

```
    unsigned long currentMillis = millis();
   struct tm timeinfo;
```

Далі виконується перевірка доступності актуального системного часу через NTP-сервер. Додатково перевіряється умова початку нової години (перша хвилина та перші секунди), що дозволяє синхронізувати періодичні події.

```
  if (getLocalTime(&timeinfo)) {
if (timeinfo.tm_min == 0 && timeinfo.tm_sec < 5) {
```

Опісля виконується запис виміряних даних у пам’ять microSD-карти. Логування виконується один раз на годину, що контролюється змінною lastLoggedHour для уникнення повторного запису.

```
 if (timeinfo.tm_hour != lastLoggedHour) {
   lastLoggedHour = timeinfo.tm_hour;
   SensorData sensor = readSensors();
   saveToSD(buildData(sensor, "sd"));
   Serial.println("Logged at exact hour");
}
```

Ця частигп коду реалізує формування та відправку щоденного звіту в Telegram. Відправка виконується у заздалегідь заданий час. Для запобігання повторної відправки протягом доби використовується змінна lastDailyDay.

```
if (timeinfo.tm_hour == dailyHour &&
```

```
    timeinfo.tm_min == dailyMinute &&
timeinfo.tm_mday != lastDailyDay) {
   lastDailyDay = timeinfo.tm_mday;
   SensorData sensor = readSensors();
   sendMessageWithMenu(CHAT_ID,
    "Daily report:\n" + buildData(sensor, "log"));
}
```

У цій частині коду реалізовано періодичний контроль параметрів навколишнього середовища. Перевірка виконується з фіксованим інтервалом часу, який задається змінною alertInterval.

```
if (currentMillis - lastAlertTime >= alertInterval) {
  lastAlertTime = currentMillis;
SensorData sensors = readSensors();
```

Після зчитування даних виконується перевірка їх валідності. У разі отримання некоректних значень (NaN) подальша обробка блоку припиняється.

```
   if (isnan(sensors.temp) || isnan(sensors.hum) || isnan(sensors.gas)) {
    Serial.println("Sensor error");
    return;
  }
```

На основі отриманих даних виконується аналіз середовища та формування попереджень (температура, вологість, газ та інші параметри).

```
      checkAlerts(sensors);
```

Цей блок забезпечує регулярне опитування Telegram API з інтервалом 1 секунда. У разі появи нових повідомлень викликається функція їх обробки.

```
 if (millis() - lastTelegramCheck >= 1000) {
   lastTelegramCheck = millis();
   handleTelegram();
 }
```

Функція handleTelegram() відповідає за взаємодію метеорологічної станції з користувачем через Telegram-бота. Вона виконує опитування серверу Telegram на наявність нових повідомлень та обробляє отримані запити.

Строка відповідає за опитування Telegram API для отримання нових повідомлень. Якщо нові повідомлення присутні, вони обробляються у циклі.

```
int numNewMessages = bot->getUpdates(bot->last_message_received + 1);
```

Кожне нове повідомлення обробляється окремо, що дозволяє підтримувати роботу з кількома запитами одночасно.

```
if(numNewMessages) {
for (int i = 0; i < numNewMessages; i++)
```

Виконується фільтрація повідомлень за ID дозволеного користувача. Далі текст повідомлення розбивається на команду та параметр.

```
String chat_id = bot->messages[i].chat_id;
String text = bot->messages[i].text;
if (chat_id != CHAT_ID) continue;
```

Також реалізовано простий парсер команд: перша частина — команда, друга — параметр (якщо є).

```
int spaceIndex = text.indexOf(' ');
```

У цьому методі також обробляються основні команди користувача:

- /start — стартове меню
- /find — пошук даних у логах SD
- /daily — встановлення часу щоденного звіту
- /mode — перемикання режиму роботи (indoor/outdoor)
- /alerts — увімкнення/вимкнення тригерів
- /calibrate — калібрування датчиків

Кожна команда змінює відповідні параметри системи або викликає функції обробки даних.

Блок відповідає за обробку натискань inline-кнопок у Telegram.

- для help надсилається список доступних команд
- для інших значень повертаються актуальні дані з датчиків

```
if (bot->messages[i].type == "callback_query")
```

Після обробки запиту формується відповідь, яка відправляється користувачу разом з інлайн-меню.

```
 if (message != "") {
   sendMessageWithMenu(chat_id, message);
 }
```

Метод sendMessageWithMenu() відповідає за формування та відправку повідомлень у Telegram з додатковим інформаційним заголовком і вбудованою клавіатурою керування.

На початку функції виконується зчитування актуальних даних з усіх датчиків, включаючи рівень заряду акумулятора. Це дозволяє формувати оновлену інформацію для кожного повідомлення.

```
String fullMessage = header + text;
```

Потім формується заголовок повідомлення, який містить індикатор заряду батареї. Він додається до кожного повідомлення для уніфікованого вигляду інтерфейсу.

```
String header = "━━━━━━ 🔋 " + String(s.bat, 0) + "% ━━━━━━\n";
```

Основний текст повідомлення об’єднується із заголовком, формуючи фінальний контент для відправки користувачу.

```
String fullMessage = header + text;
```

Формується структура inline-клавіатури Telegram у форматі JSON. Кнопки дозволяють швидко отримувати дані з різних сенсорів або викликати довідку.

```
String keyboard =
 "["
     "[{\"text\":\"All\",\"callback_data\":\"log\"}],"
     "[{\"text\":\"Temp\",\"callback_data\":\"temp\"},"
     "{\"text\":\"Hum\",\"callback_data\":\"hum\"}],"
     "[{\"text\":\"Lux\",\"callback_data\":\"lux\"},"
     "{\"text\":\"Gas\",\"callback_data\":\"gas\"}],"
     "[{\"text\":\"Mag\",\"callback_data\":\"mag\"}],"
     "[{\"text\":\"Help\",\"callback_data\":\"help\"}]"
 "]";
```

В кінці виконується відправка повідомлення користувачу разом із сформованим меню кнопок через Telegram API.

```
bot->sendMessageWithInlineKeyboard(chat_id, fullMessage, "", keyboard);
```

Функція readSensors() призначена для зчитування даних з усіх підключених датчиків та формування єдиної структури даних для подальшої обробки.

На початку функції здійснюється зчитування даних з датчика BME680. Отримуються значення температури, вологості та атмосферного тиску. Показник якості повітря визначається на основі опору газового сенсора та порівнюється з еталонним значенням, отриманим під час калібрування.

```
SensorData readSensors() {
  SensorData s;
  if (!air.performReading()) {
    Serial.println("BME680 read failed");
    s.temp = s.hum = s.gas = s.pres = NAN;
  }
  else{
    s.temp = air.temperature;
    s.hum  = air.humidity;
    s.pres = air.pressure / 100.0;
    if (gasNorm > 0) {
      float gasValue = air.gas_resistance / 1000.0;
      s.gas = (gasValue / gasNorm) * 100.0;
      s.gas = constrain(s.gas, 0, 200);
    }
    else {
      s.gas = 0;
    }
  }
}
```

Наступний блок виконує перевірку доступності магнітометра HMC5883L та зчитування його показників. До отриманих значень застосовуються поправки, визначені під час калібрування.

```
Wire.beginTransmission(0x1E);
byte errorMag = Wire.endTransmission();
if (errorMag != 0) {
  s.mX = s.mY = s.mZ = NAN;
}
else {
  sensors_event_t event;
  magnet.getEvent(&event);
  s.mX = event.magnetic.x - offsetX;
  s.mY = event.magnetic.y - offsetY;
  s.mZ = event.magnetic.z - offsetZ;
}
```

Завершальна частина функції виконує зчитування рівня освітленості з датчика BH1750 та визначає рівень заряду акумулятора. Після цього всі отримані значення повертаються у вигляді структури SensorData для подальшого використання в програмі.

```
Wire.beginTransmission(0x23);
byte errorLux = Wire.endTransmission();

if (errorLux != 0) {
  s.lux = NAN;
}
else {
  s.lux = light.readLightLevel();
}

s.bat = batteryCharge();
return s;
}
```

Функція loadConfig() призначена для завантаження конфігураційних параметрів із файлу налаштувань, розташованого на карті пам’яті microSD.

На початку функції відкривається файл конфігурації config.txt. Якщо файл відсутній або не може бути відкритий, виводиться повідомлення про помилку та виконання функції завершується.

```
void loadConfig() {
  File file = SD.open("/config.txt");
  if (!file) {
    Serial.println("Config not found");
    return;
  }
```

Далі файл зчитується построково. Кожен рядок розділяється на ключ та значення за допомогою символу «=». Це дозволяє зберігати налаштування у зручному текстовому форматі.

```
   while (file.available()) {
    String line = file.readStringUntil('\n');
    line.trim();
    if (line.length() == 0) continue;
    int separator = line.indexOf('=');
    if (separator == -1) continue;
    String key = line.substring(0, separator);
    String value = line.substring(separator + 1);
    key.trim();
   value.trim();
```

Наступний блок виконує завантаження параметрів підключення до мережі Wi-Fi та Telegram-бота.

```
    if (key == "ssid") ssid = value;
    else if (key == "password") password = value;
    else if (key == "bot_token") BOT_TOKEN = value;
    else if (key == "chat_id") CHAT_ID = value;
```

Для параметра щоденного звіту додатково виконується розбір часу у форматі «година:хвилина», після чого значення зберігаються в окремі змінні.

```
    else if (key == "daily_time") {
      int colonIndex = value.indexOf(':');
      if (colonIndex != -1) {
        dailyHour = value.substring(0, colonIndex).toInt();
        dailyMinute = value.substring(colonIndex + 1).toInt();
      }
    }
```

Окремо завантажується режим роботи метеостанції, який визначає алгоритм спрацювання тригерних повідомлень.

```
    else if (key == "mode") {
```

```
      if (value == "indoor") deviceMode = INDOOR;
      else if (value == "outdoor") deviceMode = OUTDOOR;
    }
```

Останній блок завантажує параметри калібрування газового сенсора, магнітометра та коефіцієнт корекції вимірювання напруги акумулятора.

```
    else if (key == "gas_norm") {
      gasNorm = value.toFloat();
    }
    else if (key == "offset_x") {
      offsetX = value.toFloat();
    }
    else if (key == "offset_y") {
      offsetY = value.toFloat();
    }
    else if (key == "offset_z") {
      offsetZ = value.toFloat();
    }
    else if (key == "bat_cal") {
      batCalibr = value.toFloat();
    }
```

Після завершення зчитування файл закривається, а в послідовний порт виводиться повідомлення про успішне завантаження конфігурації. У разі виявлення помилок під час зчитування система ініціалізує параметри за замовчуванням.

```
   file.close();
  Serial.println("Config loaded");
```

Функція getTime() призначена для отримання поточних дати та часу від системного годинника ESP32 і перетворення їх у текстовий формат.

На початку функція отримує поточні дату та час із системного годинника. Якщо отримати часові дані не вдалося, повертається службове значення "no_time".

Далі дата і час перетворюються у формат YYYY-MM-DD HH:MM:SS, після чого повертаються у вигляді рядка типу String. Отримане значення використовується для створення журналів вимірювань, формування звітів та відображення часу в повідомленнях Telegram.

```
String getTime() {
  struct tm timeinfo;
```

Функція buildData() призначена для формування текстового представлення показників метеостанції. Залежно від переданого параметра

type вона створює рядок для збереження на карті пам’яті або для відображення в Telegram.

Перший блок використовується для формування запису журналу на карті пам’яті microSD. Усі показники записуються в один рядок разом із датою та часом вимірювання.

```
String buildData(SensorData s, String type) {
  String time = getTime();
  if (type == "sd") {
    String result = "";
    result += time;
    result += " | Temp: " + String(s.temp, 2)+"°C";
    result += " | Hum: " + String(s.hum, 2)+"%";
    result += " | Pres: " + String(s.pres, 2)+"hPa";
    result += " | Gas: " + String(s.gas, 2)+"%";
    result += " | Lux: " + String(s.lux, 2)+"lx";
    result += " | MagXYZ: " + String(s.mX, 2) + "; "
                            + String(s.mY, 2) + "; "
                            + String(s.mZ, 2)+"uT";
    return result;
  }
```

Наступний блок створює повний звіт для Telegram. Для покращення читабельності кожен параметр виводиться з нового рядка.

```
   if (type == "log") {
    String result = "";
    result += time;
    result += "\n Temp: " + String(s.temp, 2)+"°C";
    result += "\n Hum: " + String(s.hum, 2)+"%";
    result += "\n Pres: " + String(s.pres, 2)+"hPa";
    result += "\n Gas: " + String(s.gas, 2)+"%";
    result += "\n Lux: " + String(s.lux, 2)+"lx";
    result += "\n MagXYZ: " +
              String(s.mX, 2) + "; " +
              String(s.mY, 2) + "; " +
              String(s.mZ, 2)+"uT";
    return result;
  }
```

Окремі блоки дозволяють формувати повідомлення лише для одного вибраного параметра. Ця можливість використовується кнопками Telegram-меню для швидкого перегляду потрібних даних.

```
   if (type == "temp")
    return time +
           "\n Temp: " + String(s.temp, 2)+"°C";
  if (type == "hum")
    return time +
           "\n Hum: " + String(s.hum, 2)+"%";
  if (type == "pres")
    return time +
           "\n Pres: " + String(s.pres, 2)+"hPa";
  if (type == "gas")
```

```
    return time +
           "\n Gas: " + String(s.gas, 2)+"%";
  if (type == "lux")
    return time +
           "\n Lux: " + String(s.lux, 2)+"lx";
  if (type == "mag") {
    return time +
           "\n MagXYZ: " +
           String(s.mX, 2) + "; " +
           String(s.mY, 2) + "; " +
           String(s.mZ, 2)+"uT";
   }
```

Якщо тип повідомлення не визначений, функція автоматично формує повний звіт за замовчуванням.

Метод saveToSD() призначений для збереження сформованих даних датчиків на карту пам’яті microSD. Записи автоматично сортуються за датою та додаються до відповідного файлу журналу.

Спочатку функція отримує поточний час у форматі рядка через getTime(). Якщо час недоступний, виконання переривається з повідомленням про помилку. Далі з повного рядка часу виділяється тільки дата, яка використовується для формування імені файлу журналу. Після цього відкривається (або створюється) файл у директорії /logs/ у режимі дописування, і новий запис додається в кінець файлу. У разі успішного запису виводиться підтвердження з шляхом до файлу, інакше повідомляється про помилку відкриття файлу.

```
void saveToSD(String data) {
  String fullTime = getTime();
  if (fullTime == "no_time") {
    Serial.println("Time error");
    return;
  }

  String date = fullTime.substring(0, 10);
  String path = "/logs/" + date + ".txt";
  File file = SD.open(path, FILE_APPEND);
  if (file) {
    file.println(data);
    file.close();
    Serial.println("Data logged to: " + path);
  }
  else {
    Serial.println("File error: " + path);
  }
}
```

Метод findSD() використовується для пошуку та зчитування збережених записів із карти пам’яті. Пошук може виконуватися як за датою, так і за конкретним часом запису.

На початку роботи функція аналізує параметр запиту та визначає, чи містить він лише дату, чи дату разом із часом. Після цього формується шлях до відповідного файлу журналу та виконується перевірка його наявності.

```
String findSD(String query) {
  int spaceIndex = query.indexOf(' ');
  String date;
  String time = "";
  if (spaceIndex == -1) {
    date = query;
  }
   else {
    date = query.substring(0, spaceIndex);
    time = query.substring(spaceIndex + 1);
    time.trim();
  }
  String path = "/logs/" + date + ".txt";
  if (!SD.exists(path)) return "File not found";
  File file = SD.open(path);
if (!file) return "Open error";
```

Якщо користувач вказав лише дату, функція зчитує весь вміст файлу та повертає всі записи за обраний день.

```
  if (time == "") {
    while (file.available()) {
      result += file.readStringUntil('\n') + "\n";
    }
    file.close();
    return result;
  }
```

Якщо додатково вказано час, виконується послідовний перегляд рядків файлу та пошук запису, що містить заданий часовий фрагмент. У випадку успішного пошуку повертається знайдений рядок, інакше користувач отримує повідомлення про відсутність потрібного запису.

```
  while (file.available()) {
    String line = file.readStringUntil('\n');

    if (line.indexOf(time) != -1) {
      file.close();
      return line;
    }
  }
  file.close();
  return "Not found";
}
```

Метод checkAlerts() призначений для аналізу поточних даних датчиків та формування автоматичних сповіщень у Telegram при виході параметрів за допустимі межі. Логіка сповіщень залежить від режиму роботи пристрою (в приміщенні або на вулиці) та використовує прапорці для уникнення повторних повідомлень.

Спочатку перевіряється, чи активовані сповіщення. Якщо ні — функція завершується. Далі визначається режим роботи пристрою, що впливає на текст повідомлень та порогові значення для параметрів, зокрема рівня забруднення повітря.

```
void checkAlerts(SensorData s) {
  if (!alertsEnabled) return;
  String locationTag;
  if (deviceMode == INDOOR)
    locationTag = "В будинку: ";
  else
    locationTag = "Зовні: ";
float gasLimit = (deviceMode == INDOOR) ? 80 : 75;
```

Перший блок контролює якість повітря (газ). Якщо значення виходить за допустимий поріг і повідомлення ще не надсилалось, відправляється Telegram-сповіщення. Повторне сповіщення блокується до моменту нормалізації показників.

```
  if(!gasAlertSent) {
    if(s.gas < gasLimit) {
      sendMessageWithMenu(CHAT_ID, locationTag + "Забруднене повітря");
      gasAlertSent = true;
    }
  }
  else if (s.gas >= gasLimit + 5) {
    gasAlertSent = false;
  }
```

Код нижче відповідає за контроль рівня заряду акумулятора. Сповіщення надсилається лише один раз при критичному падінні заряду та скидається після відновлення нормального рівня.

```
   if(!batAlertSent && s.bat < 15){
    sendMessageWithMenu(CHAT_ID, "Низький заряд акумулятора");
    batAlertSent = true;
  }
  else if (batAlertSent && s.bat > 20) {
    batAlertSent = false;}
```

У режимі “в приміщенні” додатково аналізуються параметри вологості та освітлення.

```
if (deviceMode == INDOOR)
```

Вологість контролюється в межах комфортного діапазону. При виході за межі надсилаються попередження, а повторні сповіщення блокуються до нормалізації значень.

```
if (!humAlertSent) {
  if (s.hum < 35) {
    sendMessageWithMenu(CHAT_ID, locationTag + "Сухе повітря");
    humAlertSent = true;
  }
  else if (s.hum > 65) {
    sendMessageWithMenu(CHAT_ID, locationTag + "Занадто вологo");
    humAlertSent = true;
  }
}
else if (s.hum >= 40 && s.hum <= 60) {
  humAlertSent = false;
}
```

Наступний блок відповідає за контроль рівня освітлення в приміщенні, з формуванням попереджень при надто низькому або надто високому значенні.

```
if (!luxAlertSent) {
  if (s.lux < 30) {
    sendMessageWithMenu(CHAT_ID, locationTag + "Низьке освітлення");
    luxAlertSent = true;
  }
  else if (s.lux > 3500) {
    sendMessageWithMenu(CHAT_ID, locationTag + "Надто яскраво");
    luxAlertSent = true;
  }
```

У режимі “зовні” логіка змінюється та додаються метеорологічні умови навколишнього середовища.

```
if (deviceMode == OUTDOOR)
```

Висока вологість на вулиці може свідчити про утворення туману, тому формується відповідне сповіщення.

```
  if (!humAlertSent) {
  if (s.hum > 90) {
   sendMessageWithMenu(CHAT_ID, locationTag + "Висока вологість можливий туман");
    humAlertSent = true;
  }
}
```

Комбінація високої вологості та зниженого атмосферного тиску використовується як індикатор можливих опадів.

```
if (s.hum > 80 && s.pres < 1005 && !rainAlertSent) {
  sendMessageWithMenu(CHAT_ID, locationTag + "Можливі опади");
  rainAlertSent = true;
}
```

Останній блок аналізує рівень освітлення для визначення погодних умов (сонячно або хмарно).

```
    if (!luxAlertSent) {
      if (s.lux > 40000) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Сонячно");
        luxAlertSent = true;
      }
      else if (s.lux < 2000 && s.lux > 500) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Хмарно");
        luxAlertSent = true;
      }
    }
```

Після нормалізації умов відповідні прапорці скидаються, що дозволяє повторно активувати сповіщення при наступному погіршенні умов.

```
   if (luxAlertSent && (s.lux <= 35000 && s.lux >= 3000)) {
    luxAlertSent = false;
  }
}
```

Метод calibrate() призначений для первинного калібрування датчиків пристрою, зокрема датчика якості повітря та магнітометра. Отримані значення використовуються для корекції подальших вимірювань і підвищення їх точності.

Спочатку виконується короткий прогрів датчика BME680 для стабілізації вимірювань перед калібруванням.

```
void calibrate() {
  Serial.println("=== CALIBRATION START ===");
  for (int i = 0; i < 10; i++) {
    air.performReading();
    delay(100);
  }
```

Далі виконується калібрування газового датчика шляхом усереднення серії вимірювань. Отримане середнє значення використовується як базовий рівень (норма) для подальшого порівняння показників.

```
   float gasSum = 0;
  int gasSamples = 30;
  for (int i = 0; i < gasSamples; i++) {
    air.performReading();
    gasSum += air.gas_resistance / 1000.0;
    delay(100);
  }
gasNorm = gasSum / gasSamples;
```

Після цього виконується калібрування магнітометра. Збирається серія вимірювань по трьох осях, після чого обчислюється середнє значення зміщення для кожної осі, яке використовується як компенсаційний офсет.

```
   float sumX = 0, sumY = 0, sumZ = 0;
  int magSamples = 50;
  for (int i = 0; i < magSamples; i++) {
    sensors_event_t event;
    magnet.getEvent(&event);
    sumX += event.magnetic.x;
    sumY += event.magnetic.y;
    sumZ += event.magnetic.z;
    delay(100); }
  offsetX = sumX / magSamples;
  offsetY = sumY / magSamples;
offsetZ = sumZ / magSamples;
```

У кінці всі обчислені калібрувальні параметри зберігаються у конфігураційний файл на SD-карті для подальшого використання після перезапуску пристрою.

```
   saveConfig("gas_norm", String(gasNorm));
  saveConfig("offset_x", String(offsetX));
  saveConfig("offset_y", String(offsetY));
  saveConfig("offset_z", String(offsetZ));
  Serial.println("Mag calibrated");
  Serial.println("CALIBRATION END");
}
```

Метод batteryCharge() використовується для оцінки рівня заряду акумулятора у відсотках на основі вимірювання напруги на аналоговому вході мікроконтролера.

Спочатку виконується усереднення кількох аналогових вимірювань для зменшення шуму та підвищення стабільності значення.

```
float batteryCharge() {
  const int samples = 10;
  float sum = 0;
  for (int i = 0; i < samples; i++) {
    sum += analogRead(BAT_PIN);
  }
  float raw = sum / samples;
```

Далі сире значення АЦП перетворюється у реальну напругу на вході мікроконтролера з урахуванням розрядності ADC та опорної напруги 3.3 В.

```
  float voltage = (raw / 4095.0) * 3.3;
```

Після цього враховується апаратний дільник напруги (2:1) та додатковий калібрувальний коефіцієнт для компенсації похибок вимірювання.

```
  voltage *= 2.0 * batCalibr;
```

Виконується обмеження діапазону напруги відповідно до типових меж Li-Ion акумулятора, де 4.2 В відповідає 100%, а 3.0 В — повному розряду.

```
  if (voltage >= 4.2) return 100;
  if (voltage <= 3.0) return 0;
```

На завершення напруга лінійно переводиться у відсотковий рівень заряду з додатковим обмеженням результату в межах 0–100%.

```
  float percent = (voltage - 3.0) * 100.0 / (4.2 - 3.0);
  return constrain(percent, 0, 100);
}
```

Повний вихідний код програмного забезпечення метеорологічної станції наведено у додатку А.

2.6 Практична реалізація проекту

Після завершення розробки апаратної та програмної частин було виконано практичну реалізацію погодної станції на базі мікроконтролера ESP32. Для перевірки працездатності розробленого пристрою було зібрано експериментальний макет, до складу якого входять мікроконтролер ESP32, датчики BME680, BH1750 та HMC5883L, модуль карти пам’яті MicroSD, модуль заряджання TP4056 та підвищувальний перетворювач напруги MT3608.

Монтаж компонентів виконано на макетній платі відповідно до розробленої схеми підключення. Такий підхід дозволив виконати налагодження апаратної частини та перевірити роботу всіх функціональних модулів системи без виготовлення друкованої плати.

Загальний вигляд експериментального макета погодної станції наведено на рисунку 2.19.

Рисунок 2.19 – Експериментальний макет погодної станції

Для наочного відображення підключення компонентів була розроблена монтажна схема в середовищі Fritzing. Схема відображає взаємозв'язки між мікроконтролером та периферійними модулями, що використовуються у складі погодної станції.

Монтажну схему пристрою наведено на рисунку 2.20.

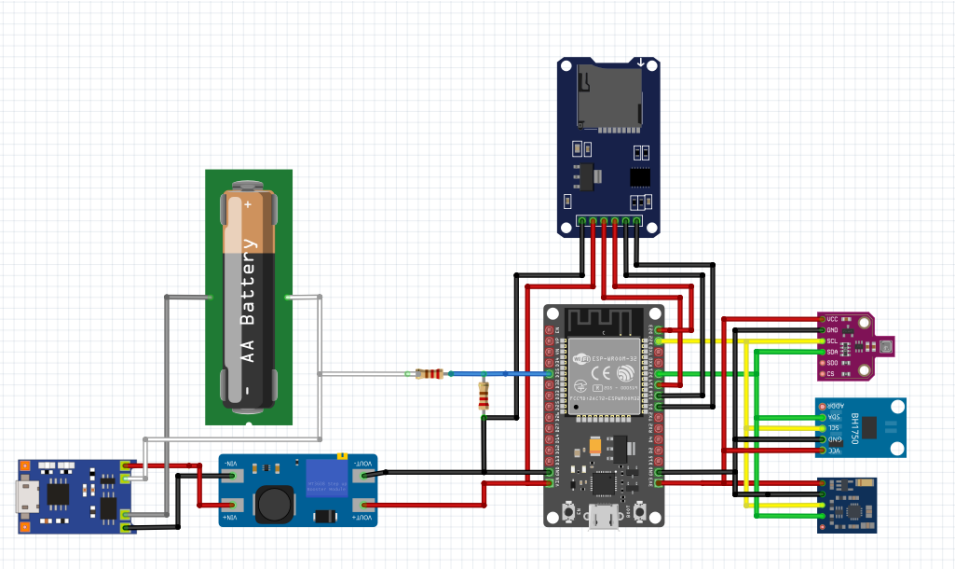


Рисунок 2.20 – Монтажна схема погодної станції, розроблена в середовищі Fritzing

Після складання макета було проведено тестування запуску системи. Під час ввімкнення контролер виконує зчитування конфігураційних параметрів, ініціалізацію датчиків, перевірку доступності карти пам'яті та підключення до бездротової мережі. Контроль процесу запуску здійснювався

за допомогою послідовного монітора Arduino IDE, у якому відображаються службові повідомлення про стан системи та результати ініціалізації модулів.

Результат запуску погодної станції наведено на рисунку 2.21.

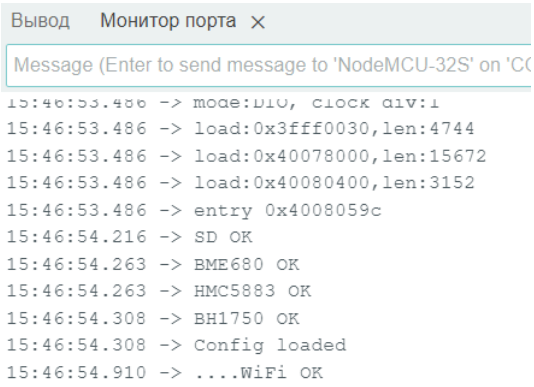


Рисунок 2.21 – Повідомлення про успішну ініціалізацію модулів у Serial Monitor

Після завершення ініціалізації пристрій переходить у робочий режим та забезпечує взаємодію з користувачем через Telegram-бот. За допомогою бота користувач може отримувати інформацію про поточний стан погодної станції, переглядати показники датчиків та використовувати передбачені програмою команди керування.

Приклад виконання команди користувача наведено на рисунку 2.22.

Рисунок 2.22 – Результат виконання команди Telegram-бота

Проведене тестування підтвердило працездатність розробленої погодної станції. У процесі перевірки було успішно виконано ініціалізацію

всіх апаратних модулів, забезпечено роботу програмного забезпечення та підтверджено коректність взаємодії пристрою з користувачем через Telegram-бот. Отримані результати свідчать про правильність обраних технічних рішень та можливість подальшого використання розробленої системи для моніторингу параметрів навколишнього середовища.

3. СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Розробка інструкції з експлуатації погодної станції

Програмне забезпечення погодної станції реалізоване на базі мікроконтролера ESP32 із використанням середовища розробки Arduino IDE. Перед початком експлуатації пристрою необхідно виконати базове налаштування апаратної та програмної частини системи.

Перед запуском пристрою користувачу необхідно підготувати карту пам’яті microSD, відформатовану у файлову систему FAT32. На карті пам’яті створюється конфігураційний файл /config.txt, який містить основні параметри роботи системи, зокрема: ім’я Wi-Fi мережі, пароль доступу, токен Telegram-бота та ідентифікатор чату.

Після підготовки програмної частини виконується встановлення пристрою та його первинний запуск. Для цього необхідно подати живлення на контролер ESP32 та дочекатися повного завантаження системи. Далі користувач через Telegram-інтерфейс надсилає команду /calibrate, яка запускає процедуру калібрування датчиків. Після завершення калібрування рекомендується перезавантажити контролер для застосування всіх збережених параметрів та коректного старту системи в робочому режимі.

У процесі подальшої експлуатації пристрій працює в автоматичному режимі, здійснює періодичний збір даних, їх збереження на карту пам’яті та передачу користувачу через Telegram-бот. Користувач має можливість отримувати як повні звіти, так і окремі параметри стану системи за допомогою команд керування.

Зокрема, через інтерфейс бота доступні запити щодо поточних показників температури, рівня освітленості та концентрації газів. Окремо реалізовано функцію дистанційної перевірки статусу підключення SD-карти та залишкового об'єму вільної пам'яті для логування. Усі отримані дані

користувач може вивантажувати у вигляді структурованого текстового звіту за визначений проміжок часу.

3.2 Розробка методики перевірки функціонування (контролю, випробування) погодної станції

У процесі експлуатації погодної станції можливе виникнення таких несправностей:

1. Відсутнє підключення до мережі Wi-Fi:
   - Перевірити наявність живлення на контролері ESP32;
   - Перевірити правильність введених параметрів мережі Wi-Fi у файлі конфігурації `/config.txt`;
   - Переконатися у працездатності точки доступу та наявності підключення до мережі Інтернет;
   - Підключити контролер до ПК та відкрити монітор послідовного порту в Arduino IDE зі швидкістю 115200 бод;
   - Перевірити повідомлення, які виводяться під час запуску пристрою, та переконатися у відсутності помилок підключення до мережі;
   - У разі необхідності перезавантажити контролер та повторити перевірку.
2. Не надходять повідомлення в Telegram:
   - Перевірити наявність доступу до мережі Інтернет;
   - Перевірити правильність токена Telegram-бота та ідентифікатора чату у файлі `/config.txt`;
   - Переконатися, що користувач раніше активував бота та не заблокував його;
   - Підключити контролер до ПК і перевірити повідомлення монітора послідовного порту на наявність помилок з’єднання;

- Якщо бот не відповідає на команди або система тривалий час не реагує, зачекати 5–10 хвилин та перезавантажити контролер;
- Після перезапуску повторно перевірити роботу Telegram-команд.

3. Не відображаються або відображаються некоректні показники датчиків:

- Перевірити правильність підключення датчиків BME680, BH1750 та HMC5883L відповідно до принципової схеми;
- Перевірити наявність живлення на всіх датчиках та надійність контактних з'єднань;
- Підключити контролер до ПК та переглянути повідомлення монітора послідовного порту щодо ініціалізації датчиків;
- Переконатися у справності ліній інтерфейсу I²C;
- Виконати повторне калібрування системи командою `/calibrate`;
- Після завершення калібрування перезавантажити контролер та перевірити показники повторно.

4. Не здійснюється запис даних на карту пам'яті:

- Перевірити правильність встановлення карти пам'яті microSD;
- Перевірити форматування карти у файловій системі FAT32;
- Перевірити наявність каталогу `/logs`;
- Підключити контролер до ПК та перевірити повідомлення монітора послідовного порту щодо ініціалізації SD-модуля;
- Перевірити справність карти пам'яті та модуля SD.

5. Не працює пошук архівних даних:

- Перевірити наявність файлів журналу у каталозі `/logs`;
- Перевірити правильність введення дати та часу у команді `/find`;
- Перевірити чи містить файл необхідні записи;

- Перевірити повідомлення монітора послідовного порту під час виконання пошуку;
- Перевірити працездатність карти пам’яті.

6. Некоректно визначається рівень заряду акумулятора:
   - Перевірити правильність підключення вимірювального дільника напруги;
   - Перевірити фактичну напругу акумулятора за допомогою мультиметра;
   - Перевірити значення коефіцієнта калібрування `bat_cal` у файлі конфігурації;
   - Переглянути показники у моніторі послідовного порту;
   - Виконати повторне калібрування системи.
7. Не надходять аварійні повідомлення:
   - Перевірити чи увімкнені сповіщення командою `/alerts on`;
   - Перевірити правильність вибраного режиму роботи командою `/mode`;
   - Переконатися, що поточні значення датчиків перевищують встановлені порогові значення;
   - Перевірити наявність підключення до мережі Інтернет;
   - Перевірити повідомлення монітора послідовного порту щодо роботи системи сповіщень;
   - Перезавантажити контролер та повторити перевірку.

4. ЕКОНОМІЧНА ЧАСТИНА

Головне завдання фінансово-економічного аналізу у складі кваліфікаційної роботи полягає в детальному кошторисному розрахунку собівартості створення системи дистанційного моніторингу погодних умов та оцінці її загальної ефективності для кінцевого споживача.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

Для розрахунку сумарного часу, витраченого на проектування та програмну реалізацію бездротової погодної станції, весь технологічний процес розробки було розділено на окремі послідовні етапи. Оцінку часових витрат для кожної операції наведено в таблиці 4.1.

Таблиця 4.1 - Середній час виконання НДР та стадії (операції) технологічного процесу

1	2	3	4
№п/п	Назва стадії	Виконавець	Середній час виконання операцій, годин
1	Аналіз технічного завдання	Програміст	2
2	Вибір елементної бази	Керівник	4
3	Розробка функціональної схеми пристрою	Програміст	2
4	Розробка Telegtam-Bot	Програміст	5

Продовження таблиці 4.1

1	2	3	4
5	Розробка алгоритму системи	Програміст	7
№п/п	Назва стадії	Виконавець	Середній час виконання операцій, годин
6	Написання текстів програми для плати ESP32	Програміст	6
7	Розробка інструкції з експлуатації електронного пристрою	Програміст	3
8	Затвердження проекту	Керівник	2
Разом			31

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Відповідно до Закону України «Про оплату праці», заробітна плата є грошовою винагородою, яку роботодавець виплачує працівникові за виконану ним роботу. Рівень оплати праці визначається складністю та умовами виконуваних завдань, професійно-кваліфікаційними якостями виконавця, кінцевими результатами його роботи, а також загальними показниками господарської діяльності підприємства.

У структурі заробітної плати виділяють основну та додаткову частини. Розрахунок основної заробітної плати ($З_{осн}$) здійснюється за формулою:

$$З_{осн}=Т_с*К_г \quad (4.1)$$

де: $Т_с$ – тарифна ставка, грн;

$К_г$ – кількість відпрацьованих годин.

Отже, основна заробітна плата для:

Програміст $З_{осн1}= 25*200=5000$ грн.

Керівника $З_{осн2}=6*300 = 1800$ грн.

Сумарна основна заробітна плата становить:

$$З_{осн.}= 5000{,}00+1800{,}00 = 6800 \text{ грн} \quad (4.2)$$

Додаткова заробітна плата становить 12% від суми основної заробітної плати:

$$З_{дод.}=З_{осн.}*К_{допл.} \quad (4.3)$$

де $К_{допл.}$ – коефіцієнт додаткових виплат працівникам: 0,12.

Отже, додаткова заробітна плата по категоріях працівників становить:

Студента $З_{дод1.} = 5000{,}00*0{,}12= 600{,}00$ грн

Керівника практики $З_{дод2.}= 1800{,}00 *0.12=216$ грн

Загальна додаткова заробітна плата становить:

$$З_{дод.}=600{,}00+216{,}00= 816{,}00 \text{ грн} \quad (4.4)$$

Звідси загальні витрати на оплату праці ($В_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.5)$$

Тому загальні витрати на оплату праці будуть становити:

$$В_{о.п.} = 6800{,}00 + 816{,}00 = 7616{,}00 \text{ грн}$$

Відрахування на соціальні заходи становлять 22%. Отже, сума відрахувань на соціальні заходи буде обчислюватися за формулою 4.6:

$$В_{с.з.} = ФОП \cdot 0{,}22 \quad (4.6)$$

де ФОП – фонд оплати праці, грн.

$$В_{с.з.} = 7616{,}00 \cdot 0{,}22 = 1\,675{,}52 \text{ грн.} \quad (4.7)$$

Розрахунки витрат на оплату праці наведено в таблиці 4.2.

Таблиця 4.2 - Зведені розрахунки витрат на оплату праці

№ п/п	Категорія прац.	Основна заробітна плата, грн			Дод.зар. плата грн	Нарахув. на ФОП, грн.	Всього витрати на оплату праці, грн
		Тариф ставка, грн	К-сть відпрац. годин	Фактично нарах. з/пл., грн.			
1	Програміст	200	25	5000,00	600,00	-	-
2	Керівник	300	6	1800,00	216,00	-	-
Разом			31	6800,00	816,00	1 675,52	9 291,52

4.3 Розрахунок матеріальних витрат

Матеріальні витрати обчислюються як добуток кількості використаних ресурсів та їхньої ринкової вартості. Загальну суму витрат на елементну базу та комплектуючі вироби визначають за формулою:

$$З_{м.в.} = \sum М_{В\,i} \quad (4.8)$$

$$З_{м.в.} = 210+400+28+66+139+119+11+25=997 \quad (4.9)$$

Проведені розрахунки занесемо у таблицю. 4.3.

Таблиця 4.3 Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Одиниця виміру	Факт. Витрачено матеріалів	Ціна одиниці, грн	Загальна сума витрат, грн
1	ESP32 NODEmcu-32s	шт.	1 шт.	210	210
2	модуль SD-карти (із AMS1117).	шт.	1 шт.	28	28
3	BME680	шт.	1 шт.	400	400
4	BH1750	шт.	1 шт.	66	66
5	HMC5883	шт.	1 шт.	138	138
6		шт.	1 шт.	119	119
7	TP4056	шт.	1 шт.	11	11
8	MT3608	шт.	1 шт.	25	25
Разом		шт.	8 шт.		997

4.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_е = W*S*T \quad (4.10)$$

де, W – необхідна потужність кВт;

T – кількість годин роботи обладання;

S – вартість кіловат-години електроенергії.

Припустимо комп`ютер споживає W = 0,6 кВт.

Вартість кіловат-години електроенергії на даний момент 4,32 грн.

$$З_е = 0,6*4,32*29 = 75,17 \text{ грн} \quad (4.11)$$

4.5 Визначення транспортних затрат

Транспортні витрати слід прогнозувати у розмірі 3–5 % від загальної суми матеріальних затрат.

$$Т_в = З_{м.в.}*0,03 \dots 0,05 \quad (4.12)$$

де $Т_в$ – транспортні втрати.

$$Т_в = 997*0,04 = 39.88 \text{ грн} \quad (4.14)$$

Отже транспортні витрати становлять 39,88 грн

4.6 Розрахунок суми амортизаційних відрахувань

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$А = \frac{Б_в * Н_а}{100\%} \quad (4.14)$$

де, А - амортизаційні відрахування за звітний період, грн.

$Б_в$ - балансова вартість групи основних фондів на початок звітного періоду, грн.;

$Н_а$ - норма амортизації, %.

Враховуючи, що ПК працює над даним проектом 29 год., тому:

$$A=\frac{40000*0,25}{150\ 100}*29=2900\text{грн} \quad (4.15)$$

4.7 Обчислення накладних витрат

Накладні витрати пов'язані з утриманням та обслуговуванням процесу розробки, забезпеченням функціонування управлінського апарату, а також створенням належних і безпечних умов праці. Залежно від організаційно-правової форми діяльності підприємства, обсяг накладних витрат може становити від 20% до 60% від суми основної та додаткової заробітної плати розробників.

Для даного проекту накладні витрати прийнято у розмірі 40% від загального фонду оплати праці. Розрахунок здійснюється за формулою:

$$Н_в=В_{о.п.}*0,2..0,6 \quad (4.16)$$

де $Н_в$– накладні витрати.

$$Н_в=7616,00*0,4=3046,40\text{грн} \quad (4.17)$$

4.8 Складання кошторису витрат та визначення собівартості НДР

Результати проведених вище розрахунків зведемо у таблиці. 4.4

Таблиця 4.4 Результати проведених розрахунків

1	2	3
Зміст витрат	Сума, грн	В % до загальної суми
Витрати на оплату праці	7616,00	46,6
Відрахування за соціальні заходи	1 675,52	10,2

Продовження таблиці 4.4

1	2	3
Матеріальні витрати	997	6,1
Витрати на електроенергію	75.17	0,5
Транспортні витрати	39,88	0,2
Амортизаційні відрахування	*2900*	*17,7*
Накладні витрати	3046,40	18,6
Собівартість	16349,968	100

Собівартість (СВ) НДР розраховується за формулою:

$$С_в = В_{о.п} + В_{с.з.} + З_{м.в.} + З_е + Т_в + А + Н_в \quad (4.18)$$

Тобто собівартість дорівнює:

$$С_в = 16349{,}968 \text{ грн} \quad (4.19)$$

4.9 Розрахунок ціни НДР

Ціну НДР визначається за формулою 4.20, що інтегрує основну та додаткову заробітну плату персоналу.

$$Ц = \frac{С_в * (1 + Р_{рен}) * К + В_{н.і.}}{К} * (1 + ПДВ) \quad (4.20)$$

де $Р_{рен}$ – рівень рентабельності;

К – кількість замовлень, од;

$В_{н.і.}$ - вартість носія інформації, грн.;

ПДВ - ставка податку на додану вартість, (20 %).

$$Ц = 16349{,}968 * (1 + 0{,}3) * (1 + 0{,}2) = 25505{,}95008 \text{грн} \quad (4.21)$$

4.10 Визначення економічної ефективнсті і терміну окупності капітальних вкладень

Економічну ефективність проекту можна визначити як результативність процесу розробки та впровадження системи, що спрямована на оптимальне використання наявних ресурсів і мінімізацію витрат при забезпеченні заданих технічних характеристик. Для оцінки доцільності інвестицій у створення бездротової погодної станції розраховують чисту теперішню вартість та термін окупності капітальних вкладень.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності ($Т_{ок}$).

$$ЧТВ= -К_в+\sum_{i=1}^{t}\frac{Г_п}{(1+i)^t} \quad (4.22)$$

де $К_в$ - затрати на проект;

$Г_п$ - грошовий потік за *t* – ий рік;

t – відповідний рік проекту;

i - величина дисконтної ставки (10…15%).

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

$$ЧТВ=-16349{,}968+\frac{9155.98}{(1+0{,}1)}+\frac{9155.98}{(1+0{,}1)^2}+\frac{9155.98}{(1+0{,}1)^3}=6419{,}60 \text{ грн} \quad (4.23)$$

Термін окупності визначається за формулою 4.24:

$$Т_{ОК}=Т_{ПВ}+\frac{Н_В}{Г_{ПР}} \quad (4.24)$$

де $Т_{ПВ}$ – період до повного відшкодування витрат, років;

$Н_В$ – невідшкодовані витрати на початок року, грн.;

$Г_{ПР}$ – грошовий потік на початок року, грн.

$$T_{ОК}=1 \ +\frac{7193,39}{9155.98}=1,79 \qquad (4.25)$$

Всі дані розрахунків внесемо в зведену таблицю 4.5 техніко-економічних показників.

Таблиця 4.5 - Техніко-економічні показники розробки мережі

№ п/п	Показник	Значення
1	Собівартість, грн	16349,968
2	Плановий прибуток, грн.	9155.98
3	Ціна, грн.	25505,95
4	Чиста теперішня вартість, грн.	6419,60
5	Термін окупності, рік	1,79

.

5. БЕЗПЕКА ЖИТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

5.1 Розрахунок системи штучного освітлення для приміщення, де здійснюються роботи з розробки погодної станції

Штучне освітлення приміщення з робочими місцями, обладнаними відеотерміналами, ЕОМ загального та персонального користування, мас бути обладнане системою загального рівномірного освітлення. У виробничих та адміністративно-громадських приміщеннях, де переважають роботи з документами, допускається вживати систему комбінованого освітлення (додатково до загального освітлення встановлюються світильники місцевого освітлення).

Загальне освітлення має бути виконане у вигляді суцільних або переривчастих ліній світильників, що розміщуються збоку від робочих місць (переважно зліва) паралельно лінії зору працівників.

Як джерело світла при штучному освітленні повинні застосовуватися, як правило, люмінесцентні лампи типу ЛБ. При обладнанні відбивного освітлення у виробничих та адміністративно-громадських приміщеннях можуть застосовуватися металогалогенові лампи потужністю до 250 Вт. Допускається у світильниках місцевого освітлення застосовувати лампи розжарювання.

Для забезпечення нормальних умов праці під час розробки та налагодження електронного пристрою (погодної станції на базі мікроконтролера ESP32) необхідно організувати достатній рівень штучного освітлення робочого місця. Якість освітлення безпосередньо впливає на точність монтажу електронних компонентів, зменшення кількості помилок при пайці та загальну продуктивність роботи.

Робоче місце належить до приміщень, де виконуються точні зорові роботи (монтаж друкованих плат, робота з дрібними елементами SMD та

DIP). Для таких умов згідно з нормами освітленості рекомендується рівень освітлення в межах 300–500 лк, а для більш точних робіт — до 750 лк.

Розраховуємо систему загального рівномірного освітлення люмінесцентними лампами для для приміщення (де здійснюються роботи з розробки погодної станції), в якому виконуються зорові роботи високої точності ($II_В$), мінімальне освітлення якого становить E = 200лм. Як світлові пристрої приймаємо світильники типу ЛПО01 (з двома лампами).

Розміри приміщення: довжина a = 7,8 м, ширина b = 7,4 м, висота h = 2,8 м. Приміщення має такі показники: коефіцієнт відбиття $\rho_{стелі}$= 70 %, $\rho_{стін}$= 50%.

Висота робочих поверхонь (столів) $h_р$ = 0,7 м. Оскільки світильники кріпляться до стелі, то їх висота над підлогою майже рівна висоті приміщення h_0 = 2,7 м.

Визначимо висоту світильника над робочою поверхнею:

$$h=h_0\text{-}h=2{,}8\text{-}0{,}7=2{,}1\text{м} \qquad (5.1)$$

Показник приміщення *i* становить:

$$i=\frac{\text{ab}}{h(a+b)}=\frac{7{,}8\cdot 7{,}4}{2{,}1\cdot(7{,}8+7{,}4)}=1{,}81 \qquad (5.2)$$

При *i* = 2 (найближче до розрахункового значення *i=1,81*) та $\rho_{стелі}$= 70 %, $\rho_{стін}$= 50% для світильника ЛПО01 коефіцієнт використання дорівнює η=54%.

Визначимо необхідну кількість світильників, для забезпечення необхідної нормованої освітленості робочих поверхонь, якщо відомо, що в кожному світильнику встановлено по дві лампи ЛБ-40, а світловий потік однієї такої лампи становить $Ф_Л$ = 3200 лм:

$$N = \frac{ESKZ}{2\Phi_{Л}\eta} = \frac{200 \cdot 57{,}52 \cdot 1{,}5 \cdot 1{,}14}{2 \cdot 3200 \cdot 0{,}54} = 5{,}69 \qquad (5.3)$$

Отже, необхідно встановити 6 світлодіодних світильників для забезпечення рівномірного освітлення робочого місця.

Світильники рекомендується розміщувати рівномірно по стелі приміщення у вигляді сітки 3×3 з урахуванням відстаней від стін для уникнення затемнених зон (див. рис. 5.1).

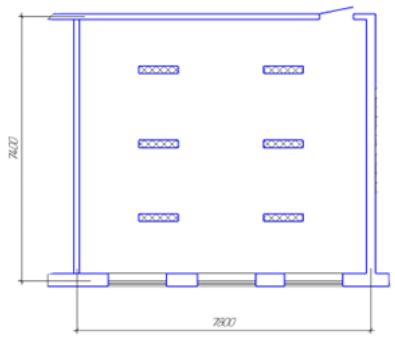


Рисунок 5.1 Схема освітлення в робочому приміщені.

5.2 Заходи попередження забруднення повітря робочої зони під час паяльно-складальних робіт при створенні апаратних платформ

Під час виготовлення та налагодження апаратної частини погодної станції виконуються паяльно-складальні роботи, пов'язані з монтажем електронних компонентів на друковані плати, підключенням датчиків та перевіркою працездатності окремих вузлів пристрою. У процесі паяння можуть утворюватися дим, аерозолі та пари флюсів, які при тривалому впливі можуть негативно впливати на органи дихання працівника.

Для забезпечення безпечних умов праці необхідно здійснювати комплекс заходів щодо попередження забруднення повітря робочої зони. Насамперед паяльні роботи повинні виконуватися в приміщеннях,

обладнаних загальнообмінною вентиляцією, яка забезпечує постійний приплив свіжого повітря та видалення забрудненого.

Для підвищення ефективності очищення повітря рекомендується використовувати місцеву витяжну вентиляцію або спеціальні відсмоктувачі диму, розташовані безпосередньо біля місця паяння. Такі пристрої дозволяють видаляти шкідливі речовини ще до їх потрапляння в зону дихання працівника.

Під час виконання монтажних робіт необхідно використовувати якісні припої та флюси, які відповідають вимогам безпеки та мають мінімальний рівень виділення шкідливих речовин. За можливості слід застосовувати безсвинцеві припої, що дозволяє знизити ризик негативного впливу на здоров'я людини та навколишнє середовище.

Робоче місце необхідно підтримувати в чистоті. Залишки припою, використаного флюсу, відрізки провідників та інші відходи повинні своєчасно збиратися та видалятися. Забороняється накопичення відходів на робочому столі, оскільки це може призводити до додаткового забруднення повітря та підвищення пожежної небезпеки.

Для зменшення концентрації шкідливих речовин у повітрі рекомендується періодично провітрювати приміщення, особливо під час тривалого виконання паяльних робіт. Також необхідно дотримуватися встановлених технологічних режимів паяння та не допускати перегріву припою і флюсу, що може призводити до інтенсивнішого утворення шкідливих випарів.

Дотримання зазначених заходів забезпечує належний стан повітряного середовища робочої зони, знижує ризик професійних захворювань та сприяє безпечним умовам праці.

ВИСНОВКИ

У кваліфікаційній роботі виконано розробку системи дистанційного моніторингу погодних умов на базі мікроконтролера ESP32. Проведено аналіз існуючих рішень у сфері метеорологічного моніторингу та обґрунтовано вибір елементної бази для створення власної погодної станції.

У процесі роботи розроблено блок схему алгоритму роботи системи, структурну схему, техніко-економічну таблицю, схему мережевох архітектури Telegram та функціональну схему пристрою, до складу якого входять датчики BME680, BH1750 та HMC5883L, модуль microSD для зберігання даних, система автономного живлення та бездротовий інтерфейс Wi-Fi. Реалізовано програмне забезпечення мовами C++ та Processing у середовищі Arduino IDE, яке забезпечує автоматичний збір, обробку та збереження інформації про параметри навколишнього середовища.

Розроблено Telegram-бот для дистанційного керування погодною станцією, перегляду поточних показників, отримання щоденних звітів, пошуку даних у журналі вимірювань та налаштування параметрів роботи системи. Також реалізовано механізм автоматичних попереджень про зміну контрольованих параметрів та систему калібрування датчиків.

У роботі виконано економічне обґрунтування проєкту, визначено собівартість розробки та проведено оцінку економічної ефективності. Розглянуто питання охорони праці під час виконання робіт з розробки та налагодження електронних пристроїв.

Отримані результати підтверджують можливість використання розробленої погодної станції для тривалого автономного моніторингу параметрів навколишнього середовища з можливістю дистанційного доступу через мережу Інтернет. Розроблена система є функціональною, масштабованою та може бути використана як основа для подальшого розширення її можливостей.

ПЕРЕЛІК ПОСИЛАНЬ

1 Методчні вказівки напрямок "Мікропроцесорні системи" [Електронний ресурс] – Режим доступу до ресурсу: https://lib.iitta.gov.ua/id/eprint/740339/2/2024%20Методичні%20рекомендації%20до%20підготовки%20кваліфікаційної%20роботи%20бакалаврів.pdf - Дата доступу: 20.06.2026.

2 Метеостанція KKMoon weather station WEA-46 [Електронний ресурс] – Режим доступу до ресурсу: https://content.rozetka.com.ua/goods/images/big/215505378.jpg - Дата доступу: 20.06.2026.

3 Характеристики KKMoon weather station WEA-46 [Електронний ресурс] –Режим доступу до ресурсу: https://bt.rozetka.com.ua/ua/310924473/p310924473/ - Дата доступу: 20.06.2026.

4 метеостанції Techno Line WS9138 [Електронний ресурс] – Режим доступу до ресурсу: https://images.prom.ua/7255731962_w640_h640_besprovodnaya-meteostantsiya-techno.jpg - Дата доступу: 20.06.2026.

5 Характеристики Techno Line WS9138 [Електронний ресурс] – Режим доступу до ресурсу: https://prom.ua/ua/p2848536783-besprovodnaya-meteostantsiya-techno.html?utm_source=google_pmax&utm_medium=cpc&utm_content=pmax&utm_campaign=Pmax_cpa_630_bytovaya_tehnika_dlya_doma_5297199152&gad_source=1&gad_campaignid=23780465516&gclid=Cj0KCQjwrs7RBhDuARIsAIVfBD34-5_AgwSrgKIrFDffuH4Xhw1OsD65lfZXteg0QFySmNQyfdCWiqgaAi3FEALw_wcB Дата доступу: 20.06.2026.

6 Wi-Fi модуль DevKit V1 з ESP-32 [Електронний ресурс] – Режим доступу до ресурсу: https://arduino.ua/prod3990-wi-fi-modyl-devkit-v1-s-esp-32?srsltid=AfmBOorLWNst9g8bvQF5Zr5t81vr97RtTVGBKpi0aqWKEw5TGJtpq3Pf- Дата доступу: 20.06.2026.

7 Модуль BME680 [Електронний ресурс] – Режим доступу до ресурсу: https://arduino.ua/ru/prod4512-modyl-datchika-kachestva-vozdyha-bme680-i2c5v?srsltid=AfmBOor0Onaf_9gxlasZ1HocLdaQue46rXhIHoTvLqLerxGOIg-YDZas - Дата доступу: 20.06.2026.

8 Модуль BH1750 [Електронний ресурс] – Режим доступу до ресурсу: https://arduino.ua/ru/prod1116-datchik-osveshhennosti-cifrovoi-bh1750fvi - Дата доступу: 20.06.2026.

9 Модуль HMC5883L [Електронний ресурс] – Режим доступу до ресурсу: https://prom.ua/ua/p63570926-magnitometr-kompas-hmc5883l.html?utm_source=google_pmax&utm_medium=cpc&utm_content=pmax&utm_campaign=Pmax_cpa_0_15_moduli_dlya_samostoyatelnoj_sborki_elektroniki_5297199152&gad_source=1&gad_campaignid=22195333185&gclid=Cj0KCQjwrs7RBhDuARIsAIVfBD1cWbzmcr6J6iARnLB9G2Q1TuSzPqAFSgcrUtgi5OcLMQu3BVS3v84aApxiEALw_wcB - Дата доступу: 20.06.2026.

10 Модуль microSD [Електронний ресурс] – Режим доступу до ресурсу: https://arduino.ua/ru/prod1601-modyl-micro-sdtf-card - Дата доступу: 20.06.2026.

11 МодульTP4056 [Електронний ресурс] – Режим доступу до ресурсу: https://arduino.ua/ru/prod4517-zaryadnii-modyl-tp4056-type-c-s-fynkciei-zashhiti-akkymylyatora - Дата доступу: 20.06.2026.

12 Перетворювач MT3608 [Електронний ресурс] – Режим доступу до ресурсу: https://arduino.ua/ru/prod1559-regyliryemii-povishaushhii-preobrazovatel-2a-28v-mt3608 - Дата доступу: 20.06.2026.

ДОДАТКИ

Додаток А. Код програми

```
#include <WiFi.h>
#include "time.h"
#include <SPI.h>
#include <SD.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME680.h>
#include <Adafruit_HMC5883_U.h>
#include <BH1750.h>

const int SD_CS = 5;
const int BAT_PIN = 35;

String ssid = "";
String password = "";

String BOT_TOKEN = "";
String CHAT_ID = "";

const char* ntpServer = "pool.ntp.org";

WiFiClientSecure client;
UniversalTelegramBot* bot;

Adafruit_BME680 air;
BH1750 light;
Adafruit_HMC5883_Unified magnet = Adafruit_HMC5883_Unified(12345);

bool gasAlertSent = false;
bool humAlertSent = false;
bool luxAlertSent = false;
bool rainAlertSent = false;
bool batAlertSent = false;
bool alertsEnabled = true;

float gasNorm = 0;
float offsetX = 0, offsetY = 0, offsetZ = 0;
float batCalibr = 1.0;

int lastLoggedHour = -1;
int lastDailyDay = -1;
int dailyHour = 7;
int dailyMinute = 0;

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

struct SensorData {
  float temp;
```

```
  float hum;
  float gas;
  float pres;
  float lux;
  float mX;
  float mY;
  float mZ;
  float bat;
};

enum Mode {
  INDOOR,
  OUTDOOR
};

Mode deviceMode = INDOOR; // по замовчуванню

SensorData readSensors();
String buildData(SensorData s, String type = "log");
void findSD(String query, String chat_id);
void saveConfig(String key, String newValue);
void saveToSD(String data);
void checkAlerts(SensorData s);
void calibrate();
float batteryCharge();

void setup() {

  Serial.begin(115200);

  Wire.begin(21, 22);
  Wire.setClock(100000);
  delay(200);

  if (!SD.begin(SD_CS)) {
    Serial.println("SD init failed");
  }
  Serial.println("SD OK");

  if (!SD.exists("/logs")) {
    SD.mkdir("/logs");
  }

  if (!air.begin(0x77)) {
    Serial.println("BME680 init failed");
  }
  else{
    air.setGasHeater(320, 150);
    Serial.println("BME680 OK");
  }

  if (!magnet.begin()) {
    Serial.println("HMC5883 init fail");
  }
  Serial.println("HMC5883 OK");

  if (!light.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x23)) {
    Serial.println("BH1750 init fail");
  }
  Serial.println("BH1750 OK");
```

```
  loadConfig();
  client.setInsecure();
  bot = new UniversalTelegramBot(BOT_TOKEN, client);

  WiFi.begin(ssid.c_str(), password.c_str());

  unsigned long wifiStart = millis();

  while (WiFi.status() != WL_CONNECTED &&
    millis() - wifiStart < 20000) {

    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi OK");

    configTime(0, 0, ntpServer);
    setenv("TZ", "EET-2EEST,M3.5.0/3,M10.5.0/4", 1);
    tzset();

    analogSetAttenuation(ADC_11db);
}

void loop() {

  unsigned long currentMillis = millis();
  struct tm timeinfo;

  if (getLocalTime(&timeinfo)) {
    if (timeinfo.tm_min == 0 && timeinfo.tm_sec < 50) {
      if (timeinfo.tm_hour != lastLoggedHour) {
        lastLoggedHour = timeinfo.tm_hour;
        SensorData sensor = readSensors();
        saveToSD(buildData(sensor, "sd"));
        Serial.println("Logged at exact hour");
      }

      if (timeinfo.tm_hour == dailyHour &&
          timeinfo.tm_min == dailyMinute &&
          timeinfo.tm_mday != lastDailyDay) {

        lastDailyDay = timeinfo.tm_mday;
        SensorData sensor = readSensors();
        sendMessageWithMenu(CHAT_ID,
          "Daily report:\n" + buildData(sensor, "log"));
      }
    }
  }

  if (alertsEnabled == true){
    if (currentMillis - lastAlertTime >= alertInterval) {
      lastAlertTime = currentMillis;
      SensorData sensors = readSensors();

      if (isnan(sensors.temp) || isnan(sensors.hum) || isnan(sensors.gas)) {
       Serial.println("Sensor error");
        return;
      }
     checkAlerts(sensors);
    }
  }
```

```
  if (millis() - lastTelegramCheck >= 1000) {
    lastTelegramCheck = millis();
    handleTelegram();
  }
}

void handleTelegram() {

  int numNewMessages = bot->getUpdates(bot->last_message_received + 1);

  if(numNewMessages) {

    for (int i = 0; i < numNewMessages; i++) {
      String chat_id = bot->messages[i].chat_id;
      String text = bot->messages[i].text;

      if (chat_id != CHAT_ID) continue;
      String message = "";
      int spaceIndex = text.indexOf(' ');
      String command;
      String value = "";

      if (spaceIndex == -1) {
        command = text;
      } else {
        command = text.substring(0, spaceIndex);
        value = text.substring(spaceIndex + 1);
        value.trim();
      }

      if (command == "/start") {
        message = "Menu";
      }

      else if (command == "/find") {

        if (value == "") {
          message = "Usage: /find YYYY-MM-DD [HH:MM]";
        }
        else {
          findSD(value, chat_id);
          message =" ";
        }
      }

      else if (command == "/daily") {
        if (value == "") {
          message = "Usage: /daily HH:MM";
        }
        else {
          int colonIndex = value.indexOf(':');

          if (colonIndex == -1) {
            message = "Wrong format!";
          }
          else {
            dailyHour = value.substring(0, colonIndex).toInt();
            dailyMinute = value.substring(colonIndex + 1).toInt();
            saveConfig("daily_time", value);
            message = "Daily set: " + value;
```

```
    }
  }
}

else if (command == "/mode") {

  if (value == "indoor") {
    deviceMode = INDOOR;
    alertsEnabled = true;
    gasAlertSent = false;
    humAlertSent = false;
    rainAlertSent = false;
    luxAlertSent = false;
    batAlertSent = false;
    saveConfig("mode", "indoor");
    message = "Режим погодної станції: \"В приміщені\" установлено.";
  }
  else if (value == "outdoor") {
    deviceMode = OUTDOOR;
    alertsEnabled = true;
    gasAlertSent = false;
    humAlertSent = false;
    rainAlertSent = false;
    luxAlertSent = false;
    batAlertSent = false;
    saveConfig("mode", "outdoor");
    message = "Режим погодної станції: \"Зовнішній\" установлено.";
  }
  else {
    message = "Usage: /mode indoor/outdoor";
  }
}

else if (command == "/alerts") {

  if (value == "on") {
    alertsEnabled = true;
    saveConfig("alert", "on");
    message = "Alerts enabled";
  }
  else if (value == "off") {
    alertsEnabled = false;
    gasAlertSent = false;
    humAlertSent = false;
    rainAlertSent = false;
    luxAlertSent = false;
    batAlertSent = false;
     saveConfig("alert", "off");
    message = "Alerts disabled";

  }
  else {
    message = "Usage: /alerts on/off";
  }

}

else if (command == "/calibrate") {
  calibrate();
  message = "Calibrate done";
}
```

```
        if (bot->messages[i].type == "callback_query") {

          bot->answerCallbackQuery(bot->messages[i].query_id);
          String callback = bot->messages[i].text;

          if (callback == "help") {
            bot->sendMessage(chat_id,
            "/find [YYYY-MM-DD] [HH:MM] - пошук логів попереднього запису.\n"
            "/daily [HH:MM] - установка щоденної події.\n"
            "/mode [indoor|outdoor] - режим погодної станції.\n"
            "/alerts [on|off] - переключатель тригерних повідомлень.\n"
            "/calibrate - калібровка значень магнітометра і газу.","");
            message =" ";
          }
          else {
            SensorData sensors = readSensors();
            message = buildData(sensors, callback);
          }
        }

        if (message != "") {
          sendMessageWithMenu(chat_id, message);
        }
      }
    }
  }

  void sendMessageWithMenu(String chat_id, String text) {
    SensorData s = readSensors();
    String header = "──────  " + String(s.bat, 0) + "% ──────\n";
    String fullMessage = header + text;
    String keyboard =
    "["
        "[{\"text\":\"All\",\"callback_data\":\"log\"}],"
        "[{\"text\":\"Temp\",\"callback_data\":\"temp\"},"
        "{\"text\":\"Hum\",\"callback_data\":\"hum\"}],"
        "[{\"text\":\"Lux\",\"callback_data\":\"lux\"},"
        "{\"text\":\"Gas\",\"callback_data\":\"gas\"}],"
        "[{\"text\":\"Mag\",\"callback_data\":\"mag\"}],"
        "[{\"text\":\"Help\",\"callback_data\":\"help\"}]"
    "]";

    bot->sendMessageWithInlineKeyboard(chat_id, fullMessage, "", keyboard);
  }

  SensorData readSensors() {
    SensorData s;
    if (!air.performReading()) {
      Serial.println("BME680 read failed");
      s.temp = s.hum = s.gas = s.pres = NAN;
    }
    else{
      s.temp = air.temperature;
      s.hum  = air.humidity;
      s.pres = air.pressure / 100.0;
      if (gasNorm > 0) {
        float gasValue = air.gas_resistance / 1000.0;
        s.gas = (gasValue / gasNorm) * 100.0;
        s.gas = constrain(s.gas, 0, 200);
      }
```

```
    else {
      s.gas = 0;
    }
  }
  Wire.beginTransmission(0x1E);
  byte errorMag = Wire.endTransmission();
  if (errorMag != 0) {
    s.mX = s.mY = s.mZ = NAN;
  }
  else {
    sensors_event_t event;
    magnet.getEvent(&event);

    s.mX = event.magnetic.x - offsetX;
    s.mY = event.magnetic.y - offsetY;
    s.mZ = event.magnetic.z - offsetZ;
  }

  Wire.beginTransmission(0x23);
  byte errorLux = Wire.endTransmission();

  if (errorLux != 0) {
    s.lux = NAN;
  }
  else {
    s.lux = light.readLightLevel();
  }

  s.bat = batteryCharge();
  return s;
}

void loadConfig() {

  File file = SD.open("/config.txt");

  if (!file) {
    Serial.println("Config not found");
    return;
  }

  while (file.available()) {
    String line = file.readStringUntil('\n');
    line.trim();

    if (line.length() == 0) continue;

    int separator = line.indexOf('=');
    if (separator == -1) continue;

    String key = line.substring(0, separator);
    String value = line.substring(separator + 1);

    key.trim();
    value.trim();

    if (key == "ssid") ssid = value;
    else if (key == "password") password = value;
    else if (key == "bot_token") BOT_TOKEN = value;
    else if (key == "chat_id") CHAT_ID = value;
    else if (key == "daily_time") {
```

```
      int colonIndex = value.indexOf(':');

      if (colonIndex != -1) {
        dailyHour = value.substring(0, colonIndex).toInt();
        dailyMinute = value.substring(colonIndex + 1).toInt();
      }
    }
    else if (key == "alert") {
      if (value == "on")  alertsEnabled = true;
      else if (value == "off")  alertsEnabled = false;
    }
    else if (key == "mode") {
      if (value == "indoor") deviceMode = INDOOR;
      else if (value == "outdoor") deviceMode = OUTDOOR;
    }
    else if (key == "gas_norm") {
      gasNorm = value.toFloat();
    }

    else if (key == "offset_x") {
      offsetX = value.toFloat();
    }
    else if (key == "offset_y") {
      offsetY = value.toFloat();
    }
    else if (key == "offset_z") {
      offsetZ = value.toFloat();
    }
    else if (key == "bat_cal") {
      batCalibr = value.toFloat();
    }

  }

  file.close();
  Serial.println("Config loaded");
}

void saveConfig(String key, String newValue) {

  File file = SD.open("/config.txt");

  if (!file) {
    Serial.println("Config read error");
    return;
  }

  File temp = SD.open("/config_tmp.txt", FILE_WRITE);
  if (!temp) {
    Serial.println("Temp file error");
    file.close();
    return;
  }

  bool found = false;
  while (file.available()) {
    String line = file.readStringUntil('\n');
    line.trim();

    if (line.length() == 0) continue;
```

```
      if (line.startsWith(key + "=")) {
        temp.println(key + "=" + newValue);
        found = true;
      } else {
        temp.println(line);
      }
    }

    if (!found) {
      temp.println(key + "=" + newValue);
    }

    file.close();
    temp.close();

    SD.remove("/config.txt");
    SD.rename("/config_tmp.txt", "/config.txt");

    Serial.println("Config updated safely: " + key);
  }

  String getTime() {
    struct tm timeinfo;

    if (!getLocalTime(&timeinfo)) {
      return "no_time";
    }

    char buffer[30];
    strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", &timeinfo);

    return String(buffer);
  }

  String buildData(SensorData s, String type) {

    String time = getTime();

    if (type == "sd") {
      String result = "";
      result += time;
      result += " | Temp: " + String(s.temp, 2)+"°C";
      result += " | Hum: " + String(s.hum, 2)+"%";
      result += " | Pres: " + String(s.pres, 2)+"hPa";
      result += " | Gas: " + String(s.gas, 2)+"%";
      result += " | Lux: " + String(s.lux, 2)+"lx";
      result += " | MagXYZ: " + String(s.mX, 2) + "; "
                               + String(s.mY, 2) + "; "
                               + String(s.mZ, 2)+"uT";
      return result;
    }

    if (type == "log") {
      String result = "";
      result += time;
      result += "\n Temp: " + String(s.temp, 2)+"°C";
      result += "\n Hum: " + String(s.hum, 2)+"%";
      result += "\n Pres: " + String(s.pres, 2)+"hPa";
      result += "\n Gas: " + String(s.gas, 2)+"%";
      result += "\n Lux: " + String(s.lux, 2)+"lx";
      result += "\n MagXYZ: " +
```

```
                String(s.mX, 2) + "; " +
                String(s.mY, 2) + "; " +
                String(s.mZ, 2)+"uT";

    return result;
  }

  if (type == "temp")
    return /*header +*/ time +
           "\n Temp: " + String(s.temp, 2)+"°C";
  if (type == "hum")
    return /*header +*/ time +
           "\n Hum: " + String(s.hum, 2)+"%";
  if (type == "pres")
    return /*header +*/ time +
           "\n Pres: " + String(s.pres, 2)+"hPa";
  if (type == "gas")
    return /*header +*/ time +
           "\n Gas: " + String(s.gas, 2)+"%";
  if (type == "lux")
    return /*header +*/ time +
           "\n Lux: " + String(s.lux, 2)+"lx";
  if (type == "mag") {
    return /*header +*/ time +
           "\n MagXYZ: " +
           String(s.mX, 2) + "; " +
           String(s.mY, 2) + "; " +
           String(s.mZ, 2)+"uT";
  }

  return buildData(s, "log");
}

void saveToSD(String data) {

  String fullTime = getTime();
  if (fullTime == "no_time") {
    Serial.println("Time error");
    return;
  }

  String date = fullTime.substring(0, 10);
  String path = "/logs/" + date + ".txt";
  File file = SD.open(path, FILE_APPEND);

  if (!file) {
    Serial.println("SD open failed: " + path);
    return;
  }

  file.println(data);
  file.close();

  Serial.println("Logged: " + path);
}

void findSD(String query, String chat_id) {

  int spaceIndex = query.indexOf(' ');

  String date;
```

```
String time = "";

if (spaceIndex == -1) {
  date = query;
}
else {
  date = query.substring(0, spaceIndex);
  time = query.substring(spaceIndex + 1);
  time.trim();
}

String path = "/logs/" + date + ".txt";

if (!SD.exists(path)){
 Serial.println("File not found");
  bot->sendMessage(chat_id, "File not found");
 return;
}

File file = SD.open(path);
if (!file){
  bot->sendMessage(chat_id, "Open error");
  Serial.println("Open error");
  return;
}

if (file.size() == 0) {
  file.close();
  bot->sendMessage(chat_id, "File empty");
  return;
}

if (time == "") {

  while (file.available()) {
   String line = file.readStringUntil('\n');

   if (line.length() > 0) {
     bot->sendMessage(chat_id, line);
     delay(150);
    }
  }

  file.close();
  return;
}

while (file.available()) {

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

  if (line.indexOf(time) != -1) {
    file.close();
    bot->sendMessage(chat_id, line);
    return;
  }
}

file.close();
bot->sendMessage(chat_id, "Not found");
return;
```

```
}

void checkAlerts(SensorData s) {

  String locationTag;

  if (deviceMode == INDOOR)
    locationTag = "В будинку: ";
  else
    locationTag = "Зовні: ";

  float gasLimit = (deviceMode == INDOOR) ? 80 : 75;

  if(!gasAlertSent) {
    if(s.gas  < gasLimit) {
      sendMessageWithMenu(CHAT_ID, locationTag + "Забруднене повітря");
      gasAlertSent = true;
    }
  }
  else if (s.gas >= gasLimit + 5) {
    gasAlertSent = false;
  }

  if(!batAlertSent && s.bat < 15){
    sendMessageWithMenu(CHAT_ID, "Низький заряд акумулятора");
        batAlertSent = true;
  }
  else if (batAlertSent && s.bat > 20) {
      batAlertSent = false;
  }

  if (deviceMode == INDOOR) {

    if (!humAlertSent) {
      if (s.hum < 35) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Сухе повітря");
        humAlertSent = true;
      }
      else if (s.hum > 65) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Занадто вологого");
        humAlertSent = true;
      }
    }
    else if (s.hum >= 40 && s.hum <= 60) {
      humAlertSent = false;
    }

    if (!luxAlertSent) {
      if (s.lux < 30) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Низьке освітлення");
        luxAlertSent = true;
      }
      else if (s.lux > 3500) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Надто яскраво");
        luxAlertSent = true;
      }
    }
    else if (luxAlertSent && s.lux >= 60 && s.lux <= 3000) {
      luxAlertSent = false;
    }
  }
```

```
  if (deviceMode == OUTDOOR) {

    if (!humAlertSent) {
      if (s.hum > 90) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Висока вологість можливий
туман");
        humAlertSent = true;
      }
    }
    else if (s.hum <= 85) {
      humAlertSent = false;
    }

    if (s.hum > 80 && s.pres < 1005 && !rainAlertSent) {
      sendMessageWithMenu(CHAT_ID, locationTag + "Можливі опади");
      rainAlertSent = true;
    }
    else if (s.hum < 75 || s.pres > 1010) {
      rainAlertSent = false;
    }

    if (!luxAlertSent) {
      if (s.lux > 40000) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Сонячно");
        luxAlertSent = true;
      }
      else if (s.lux < 2000 && s.lux > 500) {
        sendMessageWithMenu(CHAT_ID, locationTag + "Хмарно");
        luxAlertSent = true;
      }
    }

    if (luxAlertSent && (s.lux <= 35000 && s.lux >= 3000)) {
      luxAlertSent = false;
    }
  }

}

void calibrate() {

  Serial.println("=== CALIBRATION START ===");

  for (int i = 0; i < 10; i++) {
    air.performReading();
    delay(100);
  }

  float gasSum = 0;
  int gasSamples = 30;

  for (int i = 0; i < gasSamples; i++) {
    air.performReading();
    gasSum += air.gas_resistance / 1000.0; // □ кОм
    delay(100);
  }

  gasNorm = gasSum / gasSamples;

  Serial.println("Gas norm: " + String(gasNorm));
```

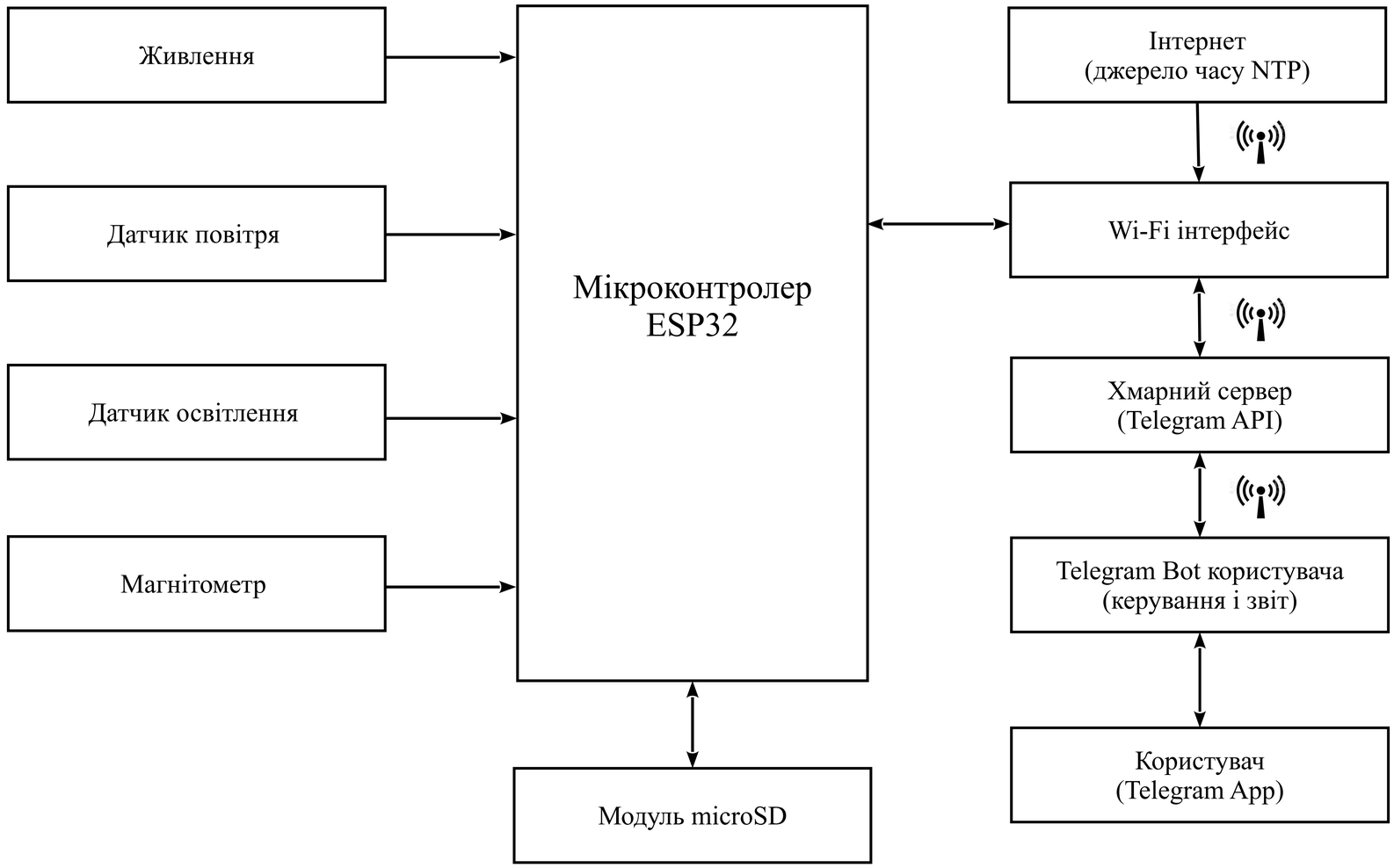
```
  float sumX = 0, sumY = 0, sumZ = 0;
  int magSamples = 50;

  for (int i = 0; i < magSamples; i++) {
    sensors_event_t event;
    magnet.getEvent(&event);

    sumX += event.magnetic.x;
    sumY += event.magnetic.y;
    sumZ += event.magnetic.z;

    delay(100);
  }

  offsetX = sumX / magSamples;
  offsetY = sumY / magSamples;
  offsetZ = sumZ / magSamples;

  saveConfig("gas_norm", String(gasNorm));
  saveConfig("offset_x", String(offsetX));
  saveConfig("offset_y", String(offsetY));
  saveConfig("offset_z", String(offsetZ));

  Serial.println("CALIBRATION END");
}

float batteryCharge() {

  const int samples = 10;
  float sum = 0;

  for (int i = 0; i < samples; i++) {
    sum += analogRead(BAT_PIN);
  }

  float raw = sum / samples;

  float voltage = (raw / 4095.0) * 3.3;

  voltage *= 2.0 * batCalibr;

  if (voltage >= 4.2) return 100;
  if (voltage <= 3.0) return 0;

  float percent = (voltage - 3.0) * 100.0 / (4.2 - 3.0);
  return constrain(percent, 0, 100);
}
```

Поз. познач.	Найменування	Кіл.	Примітка
	Аналогові модулі		.
АМ 1	Модуль TP4056, виробник Waveshare	1	
АМ2	Перетворювач MT3608, виробник Waveshare	1	
	Цифрові модулі		
DM1	Модуль microSD карти Waveshare	1	
DM2	ESP32 Devkit Wi-Fi, виробник DOIT	1	
DM3	Люксометр BH1750, виробник Waveshare	1	
DM4	Модуль BME680, виробник DFRobot	1	
DM5	Магнітометр HMC5883L, виробник Waveshare	1	
	Резистори		
R1- R2	Резистор МЛТ-0,125 100 кОм ±5%	2	
	Акумудятор		
Bat1	Li-Ion 18650 4.2V, виробник Molicel	1	
		1	

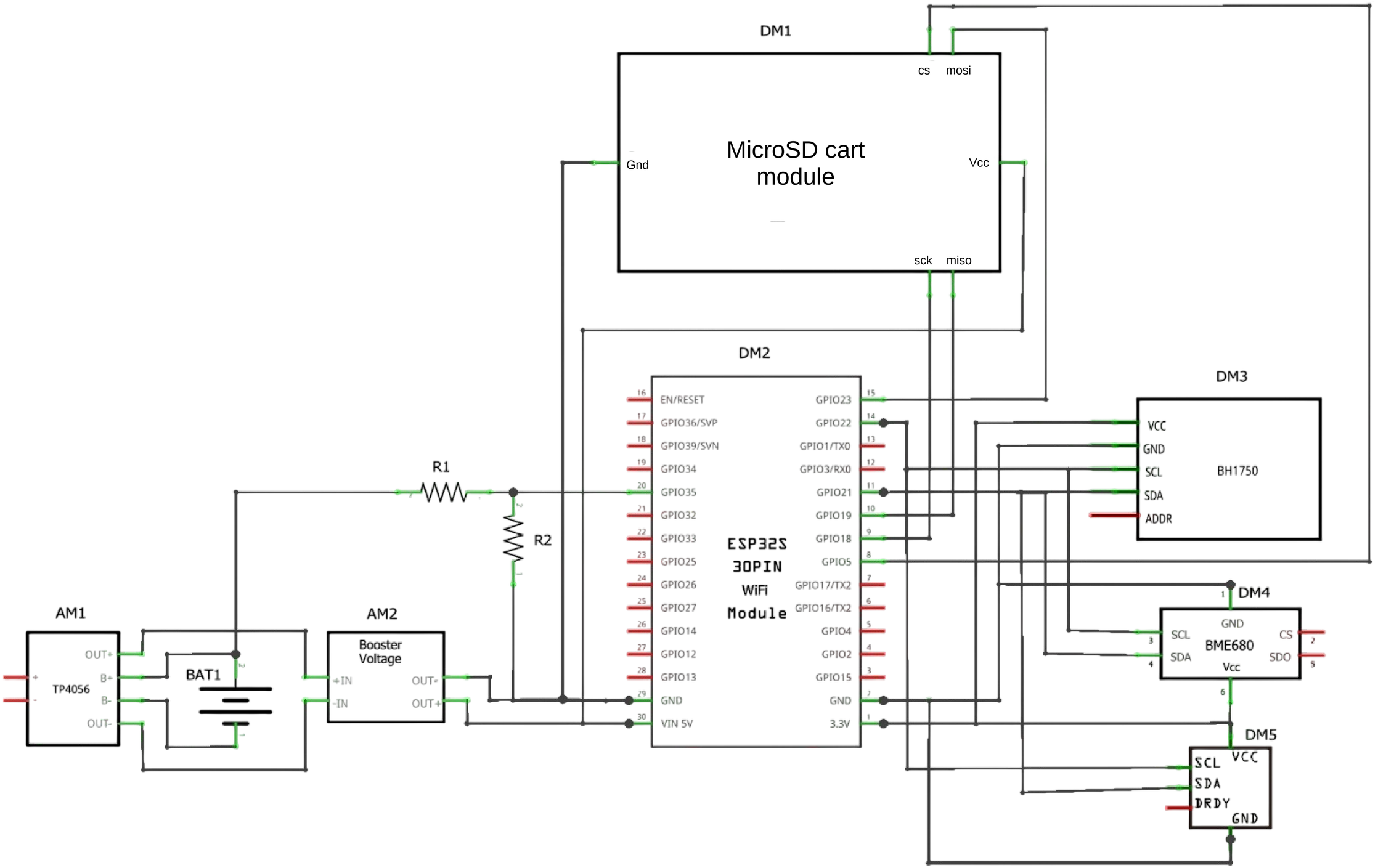
					2026.КРБ.123.703.11.00.00 ПЕ
Змн.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Рій О.В.			*Погодна станція на базі ESP32*
Перевір.		Недошитко А.Г.			
Реценз.					*Перелік елементів*
Н. Контр.		Приймак В.А.			
Затверд.					

Літ.	Арк.	Аркушів

ВСП «ТФК ТНТУ»
Група КІб-703
м. Тернопіль



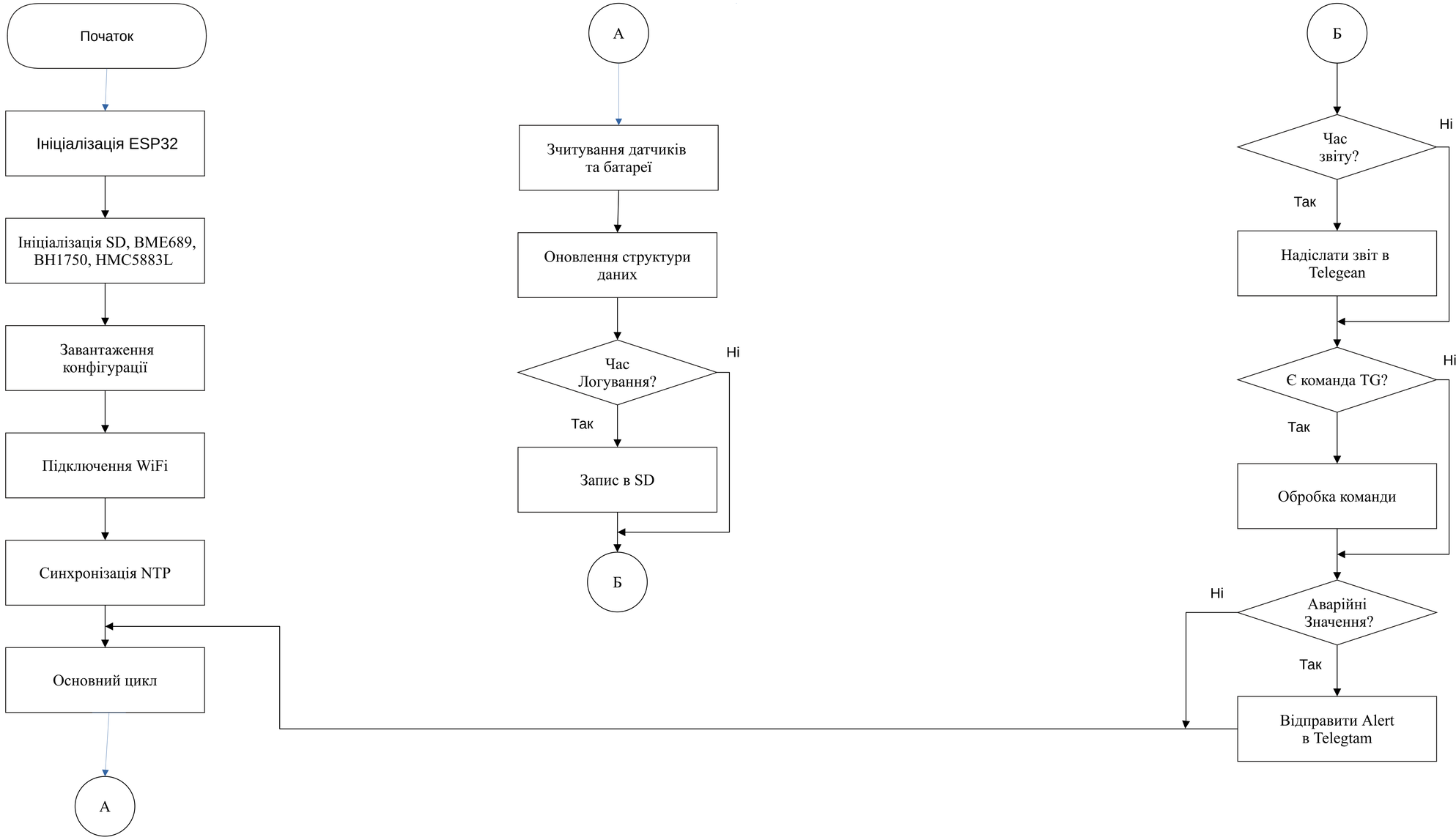
						2026.КРБ.123.703.11.00.00 Е1	
Зм.	Док.	№ докум.	Підпис.	Дата.	Розробка погодної станції на базі ESP32	Літ.	Маса	Масштаб
Розроб.		Рій О.Л.			Схема електрична структурна			
Перевір.		Недошитко А.Г.				Аркуш	Аркушів 1	
Т. конт.						ВСП "ТФК ТНТУ ім. І.Пулюя" гр.КІБ-703 м.Тернопіль		
Реценз.								
Н. конт.		Приймак В.А.						
Затв.								



					2026.КРБ.123.703.11.00.00 Е2
Зм.	Док.	№ докум.	Підпис.	Дата.	Розробка погодної станції на базі ESP32
Розроб.		Рій О.В.			Схема електрична функціональна
Перевір.		Недошитка А.Г.			
Т. конт.					
Реценз.					
Н. конт.		Приймак В.А.			
Затв.					

Лит.	Маса	Масштаб
Аркуш	Аркушів 1	

ВСП "ТФК ТНТУ ім. І.Пулюя" гр.КІБ-703 м.Тернопіль

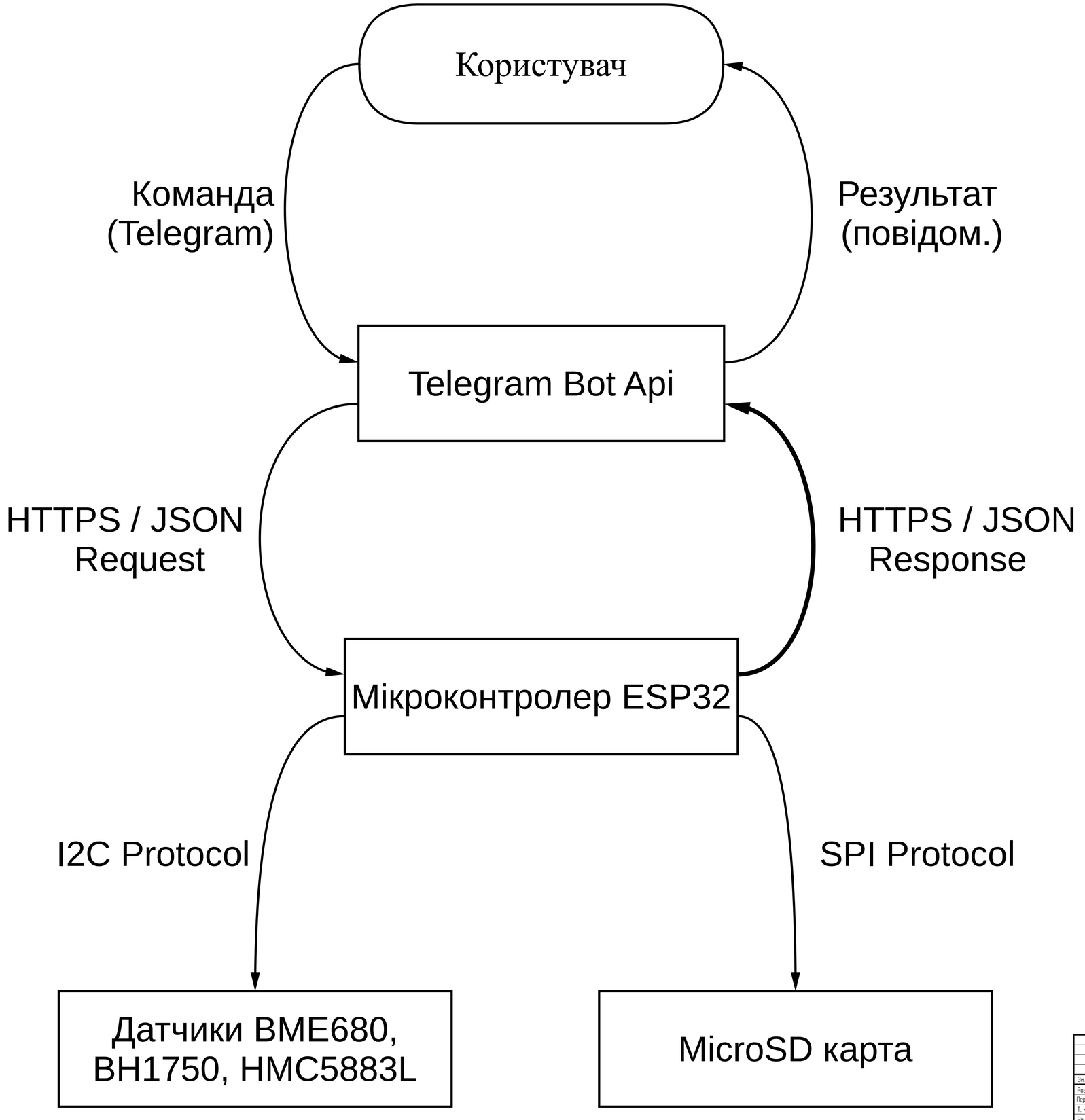


					2026.КРБ.123.703.11.00.00 БС	
Зм.	Док.	№ докум.	Підпис	Дата	Розробка погодної станції на базі ESP32 Блок-схема алгоритму роботи системи	Літ. / Маса / Масштаб
Розроб.		Рій О.В.				
Перевір.		Недошитко А.Г.				Аркуш / Аркушів 1
Т. конт.						
Реценз.						ВСП "ТФК ТНТУ ім. І.Пулюя" гр.КІБ-703 м.Тернопіль
Н. конт.		Приймак В.А.				
Затв.						

Таблиця техніко-економічних показників

Технічні показники			Економічні показники			
№ п/п	Показник	Значення	№ п/п	Показник	Одиниці вимірювання	Значення
1	Мікроконтролерні платформи	ESP-32	1	Собівартість	грн	16349,96
2	Частота роботи мережі	Wi-Fi 2.4 ГГц	2	Плановий прибуток	грн	9155.98
3	Контроль параметрів	Температура, вологість, тиск, якість повітря Уровень освітлення	3	Ціна	грн	25505,95
4	Інтерфейси	GPIO, Wi-FI, I2c, SPI, UART	4	Чиста теперішня вартість	грн	6419,60
5	Вхідні данні	Дані датчиків Команди TG Конфігупрація SD Час NTP	5	Термін окупності	рік	1,79

Зм.	Арк.	№ документа	Підпис	Дата	2026.КРБ.123.703.11.00.00 ТБ			
Розроб.		Рій О.В.			Розробка погодної станції На базі ESP32	Літ.	Маса	Масштаб
Перевір.		Недошитко А.Г.			Таблиця техніко-економічних показників			
Н.контр.						Аркуш	Аркушів	
Рецензія						ВСП "ТФК ТНТУ ім. І.Пулюя" гр.КІ-703 м.Тернопіль		
Н.контр.		Приймак В.А.						
Затв.								



					2026.КРБ.123.703.11.00.00 СА		
Зм.	Док.	№ докум.	Підпис.	Дата.	Розробка погодної станції на базі ESP32 Схема мережевої архітектури Telegram	Літ.	Маса.	Масштаб
Розроб.		Рій О.В.						
Перевір.		Недошитко А.Г.				Аркуш	Аркушів 1	
Т. конт.								
Реценз.						ВСП "ТФК ТНТУ ім. І.Пулюя" гр.КІБ-703 м.Тернопіль		
Н. конт.		Приймак В.А.						
Затв.								

МОДЕЛЬ СИСТЕМИ ПОГОДНОЇ СТАНЦІЇ