Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка
до бакалаврського дипломного проєкту
на тему:

«МІКРОКОМП'ЮТЕРНА ПОГОДНА СТАНЦІЯ ЗАСОБАМИ IoT»

Виконав: студент 4-го курсу, групи КІ-22мсз
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
_____ Морозов П.В.
(прізвище та ініціали)

Керівник: к.т.н., доц., доцент каф. ОТ
_____ Богомолов С.В.
(прізвище та ініціали)

«06» 06 2024 р.

Рецензент: к.т.н., доц., доцент каф. ПЗ
_____ Катєльніков Д.І.
(прізвище та ініціали)

«14» 06 2024 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.

«17» 06 2024 року

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.

«06» 02 2024 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКИЙ ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Морозову Павлу Володимировичу

1 Тема проєкту: «Мікрокомп'ютерна погодна станція засобами IoT», керівник Богомолов С.В. к.т.н., доц., доцент кафедри ОТ затверджені назказом вищого навчального закладу від «11» березня 2024 року №80.

2 Термін подання студентом проєкту 10.06.2024 р.

3 Вихідні дані проєкту: технічний опис датчиків DHT22 і GY-BMP280-3.3, технічний опис мікрокомп'ютера Raspberry Pi 4 Model B та електронних компонентів, опис середовища розробки Raspberry Pi OS та бібліотек pigpio, smbus2, bmp280, опис середовища розробки Flask для серверної частини та Xamarin для мобільного застосунку.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз та огляд існуючих рішень для проєктування метеорологічних станцій, поняття

метеостанцій, персональні метеостанції, дорожні метеостанції, метеостанції для аеропортів, висновки до 1-го розділу, проєктування мікрокомп'ютерної погодної станції та мобільного інтерфейсу, моделювання діаграми системи, моделювання розміщення складових частин пристрою, планування схематичної структури і функцій застосунку, висновки до 2-го розділу, огляд апаратної частини та функціональних можливостей системи, характеристика об'єкту розробки, аналіз існуючих аналогів, датчики та їх типи в метеостанціях, датчики температури, датчики тиску, мінікомп'ютер Raspberry Pi 4, висновки до 3-го підрозділу, програмна реалізація метеостанції та розробка Android-застосунку, інтеграція та налаштування датчика DHT22, інтеграція та налаштування датчика GY-BMP280-3.3, інтеграція метеостанції та зберігання даних, розробка серверу та мобільного програмного забезпечення, розробка серверу для віддаленого підключення, розробка мобільного застосунку, висновки до 4-го розділу, висновки, список використаних джерел, додатки.

5 Перелік графічного матеріалу : структурна схема підключення датчика DHT22 до мінікомп'ютера Raspberry Pi 4 Model B, структурна схема підключення датчика GY-BMP280-3.3 до мінікомп'ютера Raspberry Pi 4 Model B, структурна схема підключення датчиків DHT22 і GY-BMP280-3.3 до мінікомп'ютера Raspberry Pi 4 Model B, діаграма USE-CASE, схема алгоритму роботи застосунку.

6 Консультанти розділів проєкту наведені в таблиці 1.

Таблиця 1— Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання отримав	завдання прийняв
1,2,3	к.т.н., доц. каф. ОТ Богомолов С.В.		

7 Дата видачі завдання 12.03.2024.

8 Календарний план виконання роботи подано в таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Узгодження та затвердження теми БДП	08.03.2024	11.03.2024	вик.
2.	Аналіз літературних джерел. Попередня розробка основних розділів	12.03.2024	20.03.2024	вик.
3.	Розробка технічного завдання (ТЗ)	21.03.2024	25.03.2024	вик.
4.	Аналіз вирішення поставленої задачі на даному етапі	26.03.2024	01.04.2024	вик.
5.	Розробка структурної схеми станції	02.04.2024	07.04.2024	вик.
6.	Розробка апаратної частини та програмного забезпечення (ПЗ) для мобільних пристроїв Android	08.04.2024	10.05.2024	вик.
7.	Оформлення пояснювальної записки та графічної частини	11.05.2024	19. 05.2024	вик.
8.	Нормоконтроль	20.05.2024	20.05.2024	вик.
9.	Попередній захист БДП, доопрацювання, рецензування БДП	20.05.2024	20.05.2024	вик.
10.	Захист БДП ЕК	10.06.2024	10.06.2024	вик.

Студент _____ Морозов П.В.

Керівник проєкту _____ Богомолов С.В.

Анотація

УДК 681.004

Морозов П.В. Мікрокомп'ютерна погодна станція засобами IoT. Бакалаврський кваліфікаційний проєкт зі спеціальності 123 «Комп’ютерна інженерія», освітня програма «Комп’ютерна інженерія». Вінниця: ВНТУ, 2024. 105 с.

На укр. мові. Бібліогр.: 32 назв; рис.: 31; табл.: 1.

У бакалаврському кваліфікаційному проєкті розроблено мікрокомп'ютерну погодну станцію засобами IoT. Проєкт включає інтеграцію сенсорів DHT22 для вимірювання температури та вологості, а також GY-BMP280-3.3 для визначення атмосферного тиску та висоти над рівнем моря з мікрокомп'ютером Raspberry Pi 4 Model B. Для зчитування та обробки даних використовуються бібліотеки pigpio, smbus2 та bmp280, а для зберігання даних — реляційна база даних MariaDB.

Розроблено сервер на Flask, який з'єднується з базою даних та надає доступ до даних через REST API. Сервер підтримує mDNS для автоматичного знаходження у локальній мережі. Мобільний застосунок розроблено на Xamarin, що дозволяє отримувати та відображати дані з метеостанції у реальному часі.

Проєкт демонструє інтеграцію сучасних IoT технологій для моніторингу погодних умов, забезпечуючи точність, надійність та зручність використання.

Ключові слова: мікрокомп'ютерна система, IoT, Raspberry Pi, DHT22, GY-BMP280-3.3, Flask, MariaDB, Xamarin, метеостанція, застосунок.

Annotation

Morozov P.V. Microcomputer Weather Station Using IoT Technologies. Bachelor's Qualification Project in the field of 123 "Computer Engineering," educational program "Computer Engineering." Vinnytsia: VNTU, 2024. 105 pages.

In Ukrainian. Bibliogr.: 32 titles; figs.: 31; tables: 1.

The bachelor's qualification project develops a microcomputer weather station using IoT technologies. The project includes the integration of DHT22 sensors for measuring temperature and humidity, as well as the GY-BMP280-3.3 sensor for determining atmospheric pressure and altitude above sea level with a Raspberry Pi 4 Model B microcomputer. Libraries such as pigpio, smbus2, and bmp280 are used for data reading and processing, and the MariaDB relational database is used for data storage.

A server is developed using Flask, which connects to the database and provides data access through a REST API. The server supports mDNS for automatic discovery on the local network. A mobile application is developed using Xamarin, allowing real-time data retrieval and display from the weather station.

The project demonstrates the integration of modern IoT technologies for weather monitoring, ensuring accuracy, reliability, and ease of use.

Keywords: microcomputer system, IoT, Raspberry Pi, DHT22, GY-BMP280-3.3, Flask, MariaDB, Xamarin, weather station, application.

Зміст

Вступ

Розвиток мікроелектроніки та мікроконтролерних систем стає все більш значущим у сучасному технологічному світі, де інновації безпосередньо впливають на наше повсякденне життя та взаємодію з навколишнім середовищем. Використання Raspberry Pi для створення мікрокомп’ютерної погодної станції відкриває нові горизонти для моніторингу та аналізу погодних умов, пропонуючи детальний і точний збір даних про температуру, вологість та атмосферний тиск у реальному часі.

Актуальність теми полягає у сучасному підході до моніторингу погодних умов заснований не тільки на традиційних метеорологічних станціях, але й на використанні компактних, ефективних і доступних технологій. У численних дослідженнях і публікаціях висвітлено використання мікрокомп'ютерів та IoT-пристроїв для різних задач, включаючи метеорологічні спостереження, що демонструє зростаючий інтерес до цієї теми.

Зв’язок проєкту з науковими програмами і планами узгоджується з факультетом інформаційних технологій та комп’ютерної інженерії щодо розвитку і впровадження IoT-технологій у повсякденне життя. Проєкт відповідає тематиці досліджень кафедри обчислювальної техніки, спрямованих на інтеграцію сучасних технологій для поліпшення якості життя.

Метою дослідження є розробка та реалізація мікрокомп’ютерної погодної станції з використанням Raspberry Pi 4, яка забезпечить точне та оперативне вимірювання погодних умов. Завдання включають інтеграцію сенсорів DHT22 і GY-BMP280-3.3, розробку серверної частини на Flask, створення мобільного застосунку на Xamarin, та забезпечення віддаленого доступу до даних [1].

Об’єктом дослідження є мікрокомп’ютерна погодна станція, побудована на базі Raspberry Pi 4 Model B.

Предметом дослідження є методи і технології інтеграції сенсорів, обробки даних та їх відображення за допомогою IoT-технологій.

Методи, що використовуються у дослідженнях: моделювання,

програмування, аналіз та синтез системних компонентів, а також експериментальні методи для тестування і валідації функціональності розробленої системи.

Новизною одержаних результатів в рамках проєкту запропоновано та реалізовано інтеграцію сучасних сенсорів з мікрокомп'ютером Raspberry Pi 4, розроблено сервер на Flask та мобільний застосунок на Xamarin для моніторингу погодних умов у реальному часі, що забезпечує високу точність і надійність отриманих даних.

Публікація результатів проєкту були опубліковані в IV Міжнародній науково-теоретичній конференції «Наукові відкриття та фундаментальні наукові дослідження: світовий досвід», що підтверджує актуальність і значущість виконаного дослідження. [Електронний ресурс]. Режим доступу: https://archive.mcnd.org.ua/index.php/conference-proceeding/issue/view/07.06.2024/68

1 Огляд та аналіз існуючих рішень для проєктування метеорологічних станцій

1.1 Сучасний стан розвитку галузі метрології та метеорології

Метрологія являє собою фундаментальну основу вимірювальної техніки, яка є ключовим елементом технічного прогресу у всіх сферах людської діяльності. Основним напрямком розвитку метрології є вдосконалення теоретичних основ вимірювань, узагальнення практичного досвіду в галузі вимірювань та визначення шляхів подальшого розвитку вимірювальної техніки.

Вимірювальна техніка відіграє одну з найважливіших ролей у технічному прогресі, значно впливаючи на загальний рівень розвитку науки і техніки. Особлива увага приділяється електровимірювальній техніці, яка використовує новітні досягнення в електротехніці, електроніці, обчислювальній техніці та автоматизації для вирішення складних науково-технічних завдань.

Методи та прилади, розроблені для вимірювання електричних величин, таких як напруга, струм і опір, знайшли широке застосування не лише у своїй безпосередній сфері, а й для кількісного визначення та контролю неелектричних і магнітних параметрів. Засоби вимірювальної техніки електричних величин використовуються не лише для отримання самих вимірювальних даних, але й для моніторингу та регулювання характеристик різноманітних фізичних об'єктів і процесів. Для ефективного керування процесом вимірювання, обробки отриманих результатів та їх подальшого практичного застосування все частіше залучаються мікропроцесори, мікроконтролери та персональні комп'ютери, які забезпечують автоматизацію та оптимізацію вимірювальних операцій.

Однією з найважливіших характеристик вимірювань є точність, яка визначає міру відповідності наукового знання про досліджувані об'єкти теорії, сформульованої на основі кількісних відношень, отриманих у процесі вимірювального експерименту. Точність на кожному етапі розвитку науки і техніки залишається кінцевою метою, оскільки вона визначає достовірність і надійність отриманих даних.

Метрологія займається отриманням кількісних та якісних даних про властивості різноманітних об'єктів і процесів, а також встановленням та застосуванням наукових і організаційних принципів, технічних засобів, правил і норм, необхідних для досягнення єдності вимірювань та потрібного рівня точності. Методи метрології дозволяють експериментально перевірити достовірність отриманої вимірювальної інформації. Різноманітність джерел та способів отримання вимірювальних даних, а також сукупність методів їх обробки сприяють підвищенню точності, вірогідності та повноти даних, що, у свою чергу, поглиблює пізнання людиною властивостей матеріальних об'єктів [1].

Збільшення числа структурних елементів, програмно-апаратна реалізація засобів вимірювання й контролю призводять до їхньої якісної зміни. Вони неперервно вдосконалюються — від найпростіших мір до приладів, установок, комп'ютерно-вимірювальних систем. Використання нанотехнологій у вимірювальній техніці дозволяє досягати надзвичайно високої точності вимірювань, що відкриває нові можливості для наукових досліджень і практичних застосувань.

Сучасний розвиток метрології також пов'язаний із впровадженням нових технологій та матеріалів, які дозволяють створювати більш точні та надійні вимірювальні прилади. Інтеграція вимірювальних приладів з Інтернетом речей (IoT) сприяє створенню більш комплексних систем моніторингу та контролю, які можуть в реальному часі збирати, обробляти та аналізувати великі обсяги даних, забезпечуючи тим самим більш ефективне управління різноманітними процесами та об'єктами.

Таким чином, сучасна метрологія та метеорологія продовжують активно розвиватися, впроваджуючи нові технологічні рішення та підходи, які дозволяють досягати нових вершин у точності та надійності вимірювань, що є критично важливим для подальшого розвитку науки і техніки. Зокрема, застосування сучасних цифрових технологій, таких як штучний інтелект та машинне навчання, відкриває нові можливості для більш ефективного аналізу та обробки великих

обсягів даних вимірювань. Крім того, розробка нових вимірювальних приладів та методик забезпечує підвищення чутливості та роздільної здатності вимірювань, що дозволяє досліджувати більш тонкі та складні явища природи.

1.2 Концепції IoT, технології та застосування у сучасній науці

Інтернет речей (англ. Internet of Things, IoT) — це мережа, що об'єднує різноманітні фізичні пристрої, оснащені вбудованими датчиками та програмним забезпеченням. Ця концепція дозволяє пристроям обмінюватися даними з комп'ютерними системами, використовуючи стандартні протоколи зв'язку. Окрім датчиків для збору інформації, мережа може включати виконавчі пристрої, інтегровані в фізичні об'єкти та пов'язані між собою дротовими або бездротовими мережами. Взаємопов'язані пристрої мають здатність зчитувати дані, виконувати дії, програмуватися та ідентифікуватися. Важливою перевагою Інтернету речей є можливість функціонувати без безпосередньої участі людини завдяки використанню інтелектуальних інтерфейсів [2].

Сьогодні пристрої Інтернету речей не лише широко використовуються у повсякденному житті, але й набувають все більшого значення у сучасному бізнес-середовищі. Інтернет речей (Internet of Things або IoT) активно впроваджується у різних галузях, від промисловості до сільського господарства, рітейлу та будівництва. Пристрої IoT стають невід'ємною частиною багатьох сфер, і їх зростаюча кількість породжує нові виклики у сфері проектування, використання, обслуговування та безпеки.

Стрімкий розвиток технологій та зростання доступу до інформаційно-обчислювальних ресурсів посилюють інтерес до проблем ефективного використання мережевих ресурсів Інтернету речей і забезпечення швидкого доступу до них. IoT об'єднує фізичні об'єкти у віртуальні системи, здатні виконувати різноманітні завдання. Головна ідея полягає у з'єднанні всіх можливих об'єктів між собою та підключенні їх до мережі для збору даних і прийняття рішень на їх основі.

Наприклад, IoT може забезпечити відкриття гаражних дверей, включення кавоварки або кондиціонера, вимкнення світла тощо. Це створює нові можливості для бізнесу, охорони здоров'я, екологічної безпеки, а також змінює особисті та соціальні аспекти життя. Люди зможуть витрачати менше часу на побутові проблеми, присвячуючи більше уваги сім'ї, творчості та хобі. Підключення пристроїв до Інтернету також надає можливість більш раціонального управління ресурсами, такими як газ, вода, електроенергія (рис.1.1), що сприяє зниженню їхнього споживання та підвищенню ефективності використання [3].

Рисунок 1.1 — Використання Інтернету речей

З технологічної точки зору, IoT представляє собою мережу мереж, яка складається з унікально ідентифікованих об'єктів (так званих "речей"), що можуть взаємодіяти між собою через IP-з'єднання без втручання людини. Для передачі даних від датчиків до Інтернету необхідні дві ключові технології: маршрутизатор-шлюз і основні інтернет-протоколи, що забезпечують ефективність обміну даними.

Маршрутизатор відіграє критичну роль у питаннях безпеки, управління та направлення даних. Граничні маршрутизатори керують та контролюють стан mesh-мереж, вирівнюють та підтримують якість даних. Конфіденційність та безпека даних є також важливими аспектами, де маршрутизатор відіграє роль у створенні віртуальних приватних мереж, віртуальних локальних мереж та програмно-визначених глобальних мереж (див. рис. 1.2). Ці маршрутизатори можуть обслуговувати тисячі вузлів, що робить їх розширенням для хмари.

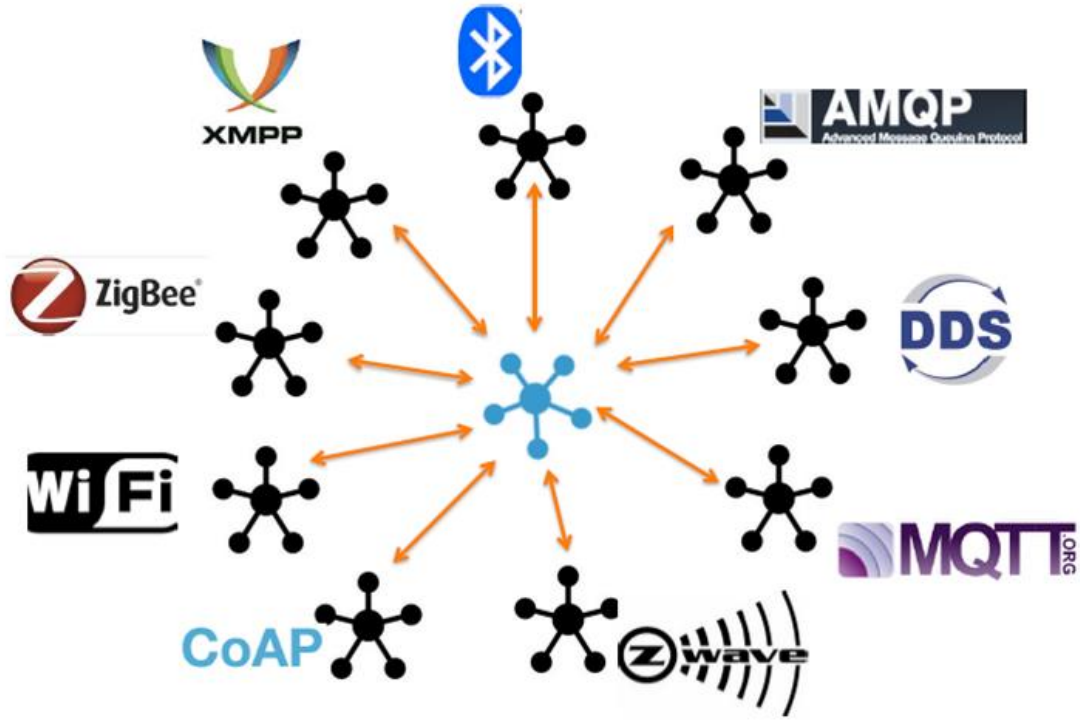

Рисунок 1.2 — Мережі та протоколи IoT

На цьому рівні використовуються різні протоколи, необхідні для обміну даними між вузлами, маршрутизаторами та хмарними сервісами у межах IoT-системи. Інтернет речей дав поштовх новим IoT-протоколам, які конкурують з традиційними протоколами HTTP та SNMP, що використовуються десятиліттями. Для передачі IoT-даних потрібні ефективні, енергозберігаючі протоколи з малою затримкою, здатні безпечно та легко відправляти дані у хмару і з неї.

Таким чином, Інтернет речей відкриває нові перспективи для розвитку різних галузей, надаючи можливість для створення розумних систем управління, що

сприяє підвищенню ефективності та безпеки у повсякденному житті та бізнесі.

1.3 Огляд існуючих аналогів метеорологічних станцій

Метеостанція — це комплекс приладів, які вимірюють атмосферні умови з метою вивчення погоди та клімату в певній місцевості. Вони є незамінними для спостереження за змінами в атмосфері та прогнозу погоди. Ці дані необхідні для дослідження та прогнозування погодних умов в певній локалізованій зоні. Стандартний набір інструментів метеостанції забезпечує моніторинг основних атмосферних показників: температури повітря, атмосферного тиску, вологості, швидкості та напрямку вітру, а також обсягу опадів. Ця інформація відіграє ключову роль у розумінні погодних процесів, а також в аналізі кліматичних змін на різних територіях [4].

Метеостанції класифікуються на офіційні та неофіційні, залежно від цілей використання зібраних даних. Офіційні метеостанції, як правило, фінансуються державними або міжнародними метеорологічними організаціями і слугують джерелом інформації для загального прогнозу погоди, досліджень у галузі клімату, а також для потреб авіації, сільського господарства та оборони. Водночас, домашні або персональні метеостанції набувають все більшої популярності серед приватних осіб, науковців-ентузіастів, а також у навчальних закладах. Вони дозволяють користувачам самостійно збирати та аналізувати метеорологічну інформацію для особистих або дослідницьких потреб, а іноді й вносити вклад у глобальні метеорологічні бази даних [5].

Персональні метеостанції — це невеликі пристрої, які використовуються для фіксації метеорологічних показників всередині приміщень та на відкритому повітрі. Зібрана ними інформація використовується для створення точного прогнозу погоди.

Сучасні технології дозволяють метеостанціям бути ще більш ефективними та точними. Ось деякі інновації:

— метеостанції тепер використовують розумні сенсори, які автоматично передають дані через Інтернет, це дозволяє віддалено моніторити погоду та отримувати оновлення в реальному часі;

— застосунки для смартфонів дозволяють користувачам отримувати погодні оновлення в будь-який час та в будь-якому місці, вони можуть включати дані з домашніх метеостанцій;

— сучасні метеостанції можуть автоматично калібрувати свої прилади, що забезпечує більш точні вимірювання.

1.3.1 Персональні метеорологічні системи

Персональні метеостанції — це невеликі пристрої, які використовуються для фіксації метеорологічних показників всередині приміщень та на відкритому повітрі (рис. 1.3). Зібрана ними інформація використовується для створення точного прогнозу погоди.

Рисунок 1.3 — Домашня персональна цифрова метеостанція

Персональні метеостанції, хоч і не досягають рівня складності та точності офіційних станцій, все ж можуть виявитися достатньо ефективними для використання у наукових дослідженнях, особливо висококласні моделі. Вони зазвичай надають інформацію про ключові погодні показники, такі як температура, вологість, атмосферний тиск, а також швидкість вітру, його напрям і обсяг опадів [6].

Деякі моделі персональних метеостанцій оснащені спеціалізованими датчиками для задоволення унікальних потреб. Наприклад, датчики блискавок та ультрафіолетове випромінювання стануть в нагоді для активних любителів відпочинку на природі, тоді як датчики вологості ґрунту і листя будуть корисні для садівників і фермерів, які прагнуть оптимізувати полив та догляд за рослинами.

Багато персональних метеостанцій мають можливість підключення до Інтернету, що дозволяє обмінюватися даними і проводити моніторинг погодних умов дистанційно через такі сервіси, як Weather Underground і Ambient Weather Network. Ця функція також дозволяє інтегрувати погодні дані з системами розумного будинку, автоматизуючи реакцію на зміну погодних умов.

Дані з персональних метеостанцій важливі не лише для індивідуального використання. Метеорологи та дослідники активно використовують цю інформацію, особливо в регіонах, де відсутні офіційні метеостанції, допомагаючи заповнити існуючі прогалини у погодних даних. Програма спільних погодних спостережень служить яскравим прикладом такої співпраці, де волонтери діляться даними про погоду з Національною метеорологічною службою, збагачуючи загальну базу даних та підвищуючи точність прогнозів.

1.3.2 Автомобільні метеорологічні комплекси

Сильні вітри, град та інші несприятливі погодні явища можуть мати значний вплив на транспортний рух. Вітер здатний завдавати серйозної шкоди будівлям, рухомим об'єктам тощо. Він може пошкоджувати залізничні колії, мости, комунікації та енергетичні об'єкти, повалювати дерева на дороги, що призводить

до перекриття руху транспорту, здіймати пісок, погіршуючи видимість. Вітер також може безпосередньо впливати на рух потягів, сповільнюючи їхню швидкість або навіть призводячи до сходження з рейок. Для прогнозування таких небезпечних погодних умов використовуються дорожні метеостанції.

У контексті забезпечення безпеки дорожнього руху, дорожні метеостанції прагнуть дотримуватися найвищих стандартів якості. Завдяки детальним вимірюванням таких параметрів довкілля, як температура поверхні дороги, товщина водної плівки, температура замерзання та низькі температури, придорожні метеостанції дозволяють відстежувати місцеві погодні умови. Це забезпечує постійне надходження точної інформації про стан доріг.

Інтелектуальні дорожні датчики встановлюються на рівні дороги або над нею, і щохвилини повідомляють про її стан: сухий, вологий, слизький чи крижаний, або вкритий снігом. Зображення типової дорожньої метеостанції представлено на рисунку 1.4.

Рисунок 1.4 — Дорожня метеостанція

Придорожні метеостанції можуть надавати в реальному часі критично важливі дані, такі як температура та вологість повітря, тип та кількість опадів,

напрямок та швидкість вітру, атмосферний тиск та видимість. Кожна станція складається з інтелектуального дорожнього датчика, компактної метеостанції та кольорової камери. Система збору даних передає не лише дані, але й кольорові фотографії з окремих станцій зі швидкістю кількох кадрів на секунду, якщо це потрібно. У багатьох галузях промисловості доступ до гідрологічних даних є життєво необхідним. Незалежно від того, чи йдеться про проєктування зрошувальних систем, забезпечення безпеки питної води чи прогнозування та запобігання повеням та посухам, знання місцевих погодних умов є критичним. Тому дані про опади мають вирішальне значення для гідрологічних моделей, що використовуються для управління водними ресурсами, прогнозування та моніторингу повеней і посух у різних масштабах та місцях, таких як водосховища, гідроелектростанції, підприємства водопідготовки та водовідведення. Збір даних, що керує процесами прогнозування, нагляду та контролю за гідроенергетичними операціями, відбувається на гідрологічних, річкових та рівнях опадів, за допомогою наземних спостережень дощомірами, а також на метеорологічних станціях. Частина даних отримується за допомогою автоматичних станцій, які зчитують інформацію з датчиків і передають її на електростанцію за допомогою супутника та Інтернету [7].

1.3.3 Авіаційні метеорологічні станції

Аеропортові метеостанції є віддаленими автоматизованими системами моніторингу погоди, призначеними для обслуговування авіаційних та метеорологічних операцій, а також прогнозування погодних умов. Типова аеропортова метеостанція зображена на рисунку 1.5. Основними параметрами, що вимірюються такими станціями, є швидкість та напрямок вітру, атмосферний тиск, опади (дощ та сніг), освітленість, температура і вологість повітря, сонячна радіація, концентрація твердих частинок.

Автоматична метеостанція аеропорту розроблена та впроваджена з використанням новітніх сенсорних технологій та сучасних засобів зв'язку. Вона має широкі функціональні можливості, включаючи калібрування, виявлення

несправностей, управління статусом, гарантію живлення, блискавкозахист та заходи безпеки.

Рисунок 1.5 — Метеостанція аеропорту AWOS/RVR

Це повністю налаштована система моніторингу погоди в аеропорту, яка може забезпечувати безперервний потік даних в режимі реального часу для пілотів, керівників аеропортів та менеджерів, а також генерувати звіти.

Автоматизовані метеостанції відіграють важливу роль у забезпеченні безпеки та ефективності авіаційних операцій. Точні та своєчасні дані про погодні умови, отримані з аеропортових метеостанцій, допомагають пілотам приймати обґрунтовані рішення щодо зльоту, посадки та маршруту польоту. Крім того, ця інформація є критичною для планування та координації робіт на аеродромі, таких як розчистка злітно-посадкових смуг від снігу чи льоду, а також для прогнозування можливих затримок чи скасувань рейсів через несприятливі погодні умови.

Метеостанції критично важливі для моніторингу атмосферних умов, вони дозволяють науковцям, державним організаціям та приватним особам отримувати необхідні дані для аналізу та прогнозування погоди. Завдяки сучасним технологіям, такі станції стають ще більш ефективними і точними, надаючи можливість віддаленого доступу до даних в реальному часі.

2 Розробка мікрокомп'ютерної погодної станції: апаратна частина та мобільний інтерфейс

2.1 Моделювання архітектури системи та функціональних діаграм

UML — це стандартизована мова візуального моделювання, що застосовується в об'єктно-орієнтованому програмуванні. Вона є невід'ємним компонентом уніфікованого процесу розробки програмного забезпечення. UML — це універсальна мова широкого спектру, яка є відкритим стандартом та використовує графічну нотацію для створення абстрактної моделі системи, відомої як UML-модель.

UML була розроблена з метою забезпечення чіткого визначення, наочної візуалізації, ретельного проєктування та всебічної документації, в першу чергу, програмних систем. UML не є мовою програмування у прямому сенсі, оскільки не призначена для безпосереднього написання виконуваного коду. Проте, в середовищах, які підтримують виконання UML-моделей як інтерпретованих специфікацій, існує можливість автоматичної генерації програмного коду на основі створених UML-діаграм та описів за допомогою спеціальних інструментів кодогенерації.

Універсальна мова моделювання UML призначена для використання на всіх етапах життєвого циклу аналізу бізнес-проєктів та розробки прикладних програмних систем. Завдяки різноманітним типам діаграм, що підтримуються нотацією UML, і широкому спектру можливостей для представлення різних аспектів системи, ця мова стала універсальним інструментом для опису як програмних, так і бізнес-орієнтованих систем. Багатогранність UML дозволяє застосовувати її для моделювання, візуалізації та документування в рамках аналітичних і проєктувальних робіт з розвитку застосунків та бізнес-процесів.

Діаграма прецедентів (або діаграма варіантів використання) в нотації UML є графічним представленням взаємовідносин між акторами (зовнішніми сутностями, що взаємодіють із системою) та прецедентами (варіантами використання, що

описують функціональні можливості системи). Ця діаграма являє собою граф, що містить множину акторів, прецедентів, обмежених межами системи (прямокутником), асоціацій між акторами та прецедентами, відношень між прецедентами, а також відношень узагальнення для акторів. Діаграми прецедентів візуалізують елементи моделі варіантів використання, надаючи загальне уявлення про функціональність розроблюваної системи та способи її використання різними зовнішніми сутностями.

Основна ідея діаграми прецедентів (або діаграми варіантів використання) полягає в тому, що проєктована система представляється як сукупність сутностей, або акторів, які взаємодіють з системою через певні варіанти використання.

Іншими словами, дана діаграма візуалізує розроблювану систему як набір функціональних можливостей (варіантів використання), з якими взаємодіють різні зовнішні сутності (актори), такі як користувачі, інші системи або зовнішні пристрої.

Варіант використання (USE-CASE) є засобом специфікації функціональних можливостей системи, які вона надає зовнішнім сутностям, званим акторами. Кожен варіант використання описує набір дій чи послідовність кроків, які система виконує в процесі взаємодії з актором. При цьому варіант використання не деталізує спосіб реалізації такої взаємодії між акторами та системою, а лише визначає очікувану поведінку системи на концептуальному рівні [8].

Для візуалізації функціональних вимог та способів взаємодії зовнішніх сутностей з розроблюваною системою використовується діаграма прецедентів (або діаграма варіантів використання). Графічне представлення цієї діаграми для даного об'єкта розробки матиме вигляд (рисунок 2.1).

Діаграма діяльності в UML є графічним поданням графа діяльностей. Граф діяльностей є різновидом графа кінцевого автомата станів, де вузлами виступають певні дії, а переходи між ними відбуваються після завершення виконання цих дій.

Дія є базовим елементом специфікації поведінки.

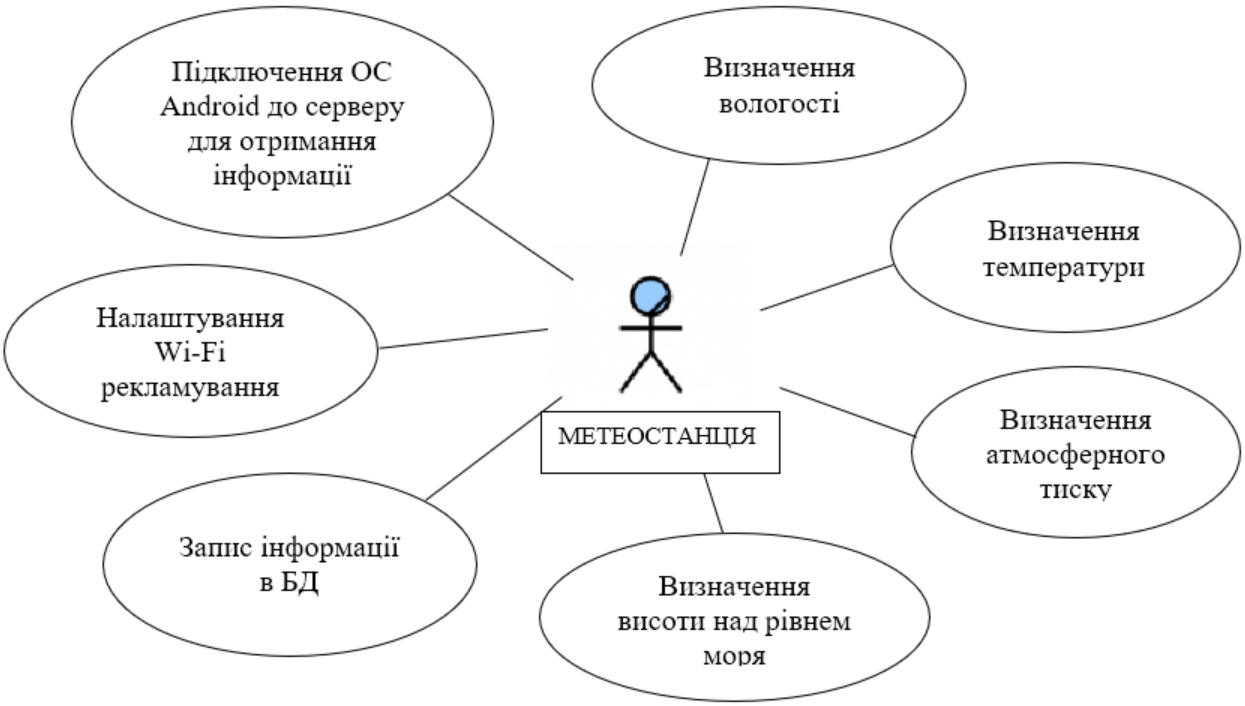

Рисунок 2.1 — Діаграма USE-CASE

Вона приймає множину вхідних сигналів та перетворює їх на множину вихідних сигналів. При цьому одна або обидві множини можуть бути порожніми. Виконання дії відповідає виконанню окремої елементарної операції. Аналогічно, виконання діяльності полягає у виконанні сукупності дій, що входять до її складу. В процесі одного виконання діяльності кожна дія в її складі може бути виконана один або кілька разів.

Як мінімум, дії повинні передбачати отримання вхідних даних, їх перетворення та перевірку. Деякі дії можуть вимагати певної послідовності виконання. Специфікація діяльності (на вищих рівнях абстракції) може дозволяти одночасне виконання декількох логічних потоків та містити механізми синхронізації для забезпечення коректного порядку виконання дій.

Ці діаграми активно застосовуються для опису поведінки, яка включає численні паралельні процеси. Кожен стан у такій діаграмі активності відображає виконання окремої базової операції, а перехід до наступного стану відбувається лише після того, як операція буде завершена.

Отже, діаграму активності можна розглядати як специфічний варіант діаграми станів.

Основна мета застосування діаграми активності полягає у візуалізації особливостей виконання операцій класів, коли виникає потреба демонструвати алгоритми їх реалізації.

Основні кроки розробки метеостанції відображені на рисунку 2.2 [9]:

Рисунок 2.2 — Діаграма діяльності розробки метеостанції

2.2 Моделювання розміщення складових частин пристрою

Розробка схеми метеостанції відбуватиметься покроково, поступово інтегруючи необхідні компоненти (піни) та створюючи окремі програмні скрипти для їх функціонування. Такий підхід дозволить ефективно протестувати роботу всіх елементів ще на етапі проєктування.

Після успішного проходження тестів усі окремі програмні скрипти будуть інтегровані в один цілісний проєкт.

Першочергово необхідно розробити схему інтеграції датчика GY-BMP280-3.3 для вимірювання атмосферного тиску та висоти над рівнем моря з обраним мінікомп'ютером (Raspberry Pi 4 Model B). Оскільки цей компонент відсутній у стандартній бібліотеці Fritzing, його потрібно завантажити додатково з інтернет-ресурсу [10, 11].

На рисунку 2.3 наведено приклад розташування схеми на мінікомп'ютері, проте це не є обов'язковою конфігурацією, оскільки при додаванні нових елементів до схеми їх розміщення може бути змінено.

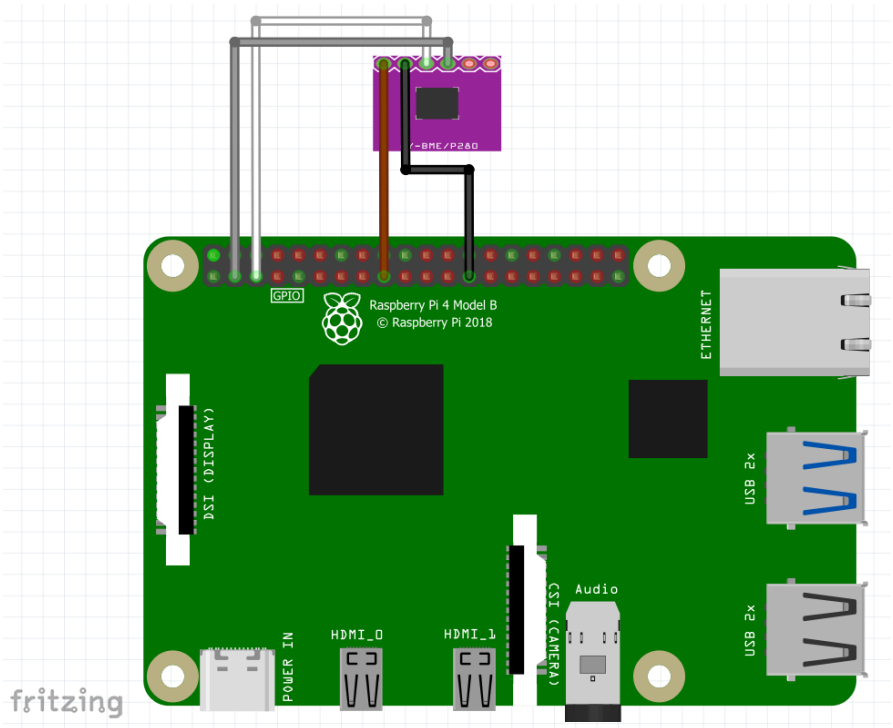

Рис. 2.3 — Схема підключення датчику GY-BMP280-3.3 до мінікомп’ютера Raspberry Pi 4 Model B

Наступним кроком буде створення схеми у програмі Fritzing для інтеграції датчика DHT22 (який також потрібно завантажити додатково) з метою вимірювання рівня вологості та температури (рисунок 2.4) [11].

Після поєднання всіх компонентів в єдину схему, ми отримаємо результат, зображений на рисунку 2.5 [11].

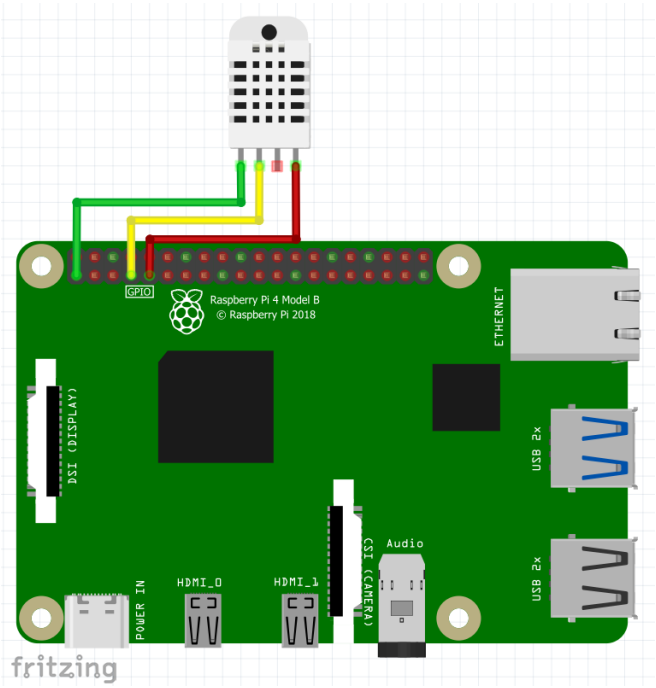

Рисунок 2.4 — Схема підключення датчику DHT22 до мінікомп’ютера Raspberry Pi 4 Model B

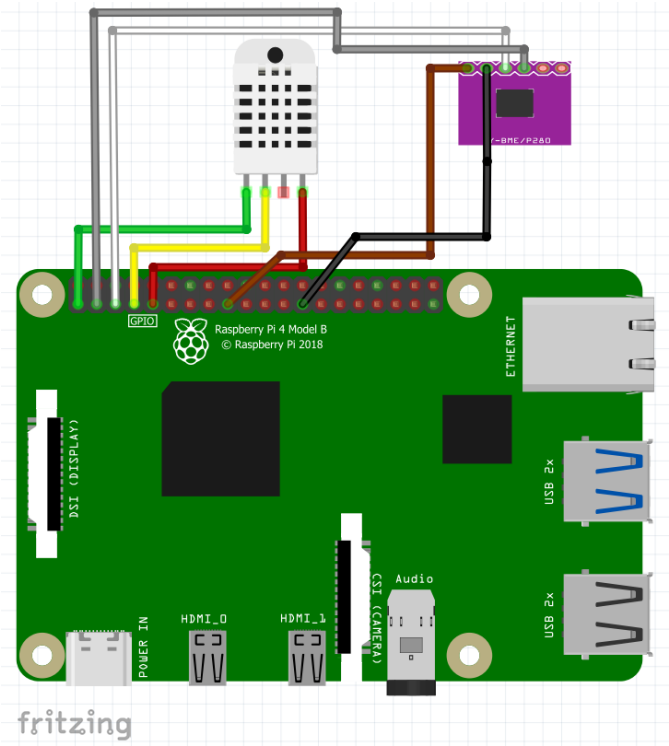

Рисунок 2.5 — Схема з підключеними датчиками для метеостанції

Схема демонструє підключення датчиків DHT22 і GY-BMP280-3.3 до мінікомп’ютера Raspberry Pi 4 Model B.

Схема забезпечує з'єднання датчиків з мінікомп'ютером таким чином, що вони можуть надсилати зібрані метеорологічні дані на Raspberry Pi 4, де ці дані можуть бути оброблені та використані для моніторингу погодних умов.

2.3 Планування схематичної структури і функцій застосунку

У рамках розробки мобільних застосунків для операційної системи Android кожен застосунок функціонує за допомогою спеціальних компонентів, що називаються «activity». Фактично, кожен екран чи сторінка застосунку для Android представлений окремим компонентом activity, і кожен такий компонент має власний життєвий цикл. Платформа Xamarin.Android, яка дозволяє створювати застосунки для Android за допомогою мови програмування C#, також підтримує цю концепцію activity та надає можливість описувати їх використовуючи C#. Всі компоненти activity в Xamarin.Android позначаються спеціальними атрибутами, що дозволяє інтегрувати їх в Android Manifest — файл, де визначаються та включаються основні складові застосунку. Для кращого розуміння розглянемо детально етапи життєвого циклу компонентів activity (рисунок 2.6) [12].

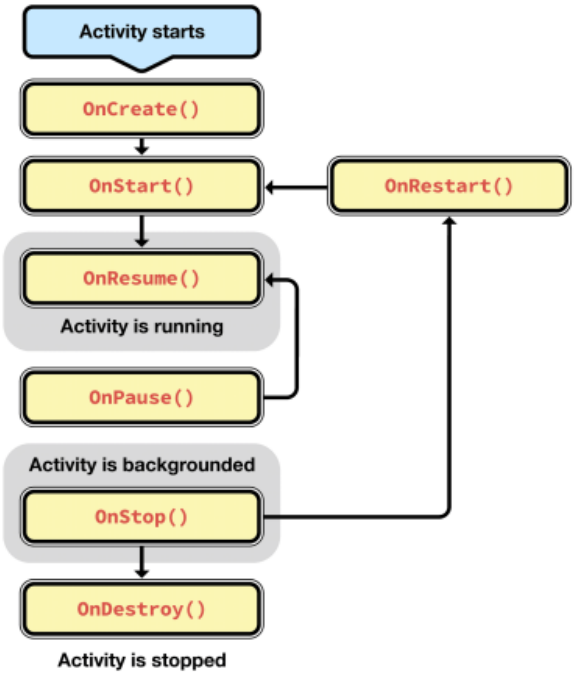

Рисунок 2.6 — Методи життєвого циклу Android

Коли застосунок запускається, спочатку викликається метод OnCreate. У цьому методі можна відновити раніше збережені дані, наприклад, якщо застосунок був перезапущений;

Після цього викликається метод OnStart, який необхідний для налаштування візуальних елементів (View) — тобто ініціалізації користувацького інтерфейсу;

Метод OnResume завжди викликається після OnStart. Тут можна ініціалізувати камеру, або інші ресурси, потрібні для коректної роботи застосунку;

OnPause викликається, коли система переводить дане activity у фоновий режим або коли воно перекривається іншим activity. У цьому методі рекомендується зберегти всі необхідні дані та звільнити ресурси, що використовуються застосунком;

OnStop викликається, коли користувач більше не бачить поточне activity. Краще не використовувати цей метод для очищення та збереження ресурсів, а робити це у OnPause;

OnDestroy викликається, коли activity видаляється з пам'яті пристрою. Проте, в деяких ситуаціях ОС Android може не викликати цей метод, тому покладатися на нього не безпечно;

OnRestart викликається після OnStop, якщо користувач знову активує дане activity.

Перед розробкою мобільного застосунку для моніторингу погодних даних з мікрокомп'ютерної метеостанції розроблено чіткий визначений алгоритм роботи, який візуалізовано на блок-схемі (див. рисунок 2.7). Алгоритм описує послідовність дій та переходів між різними станами програми під час її виконання, враховуючи можливі варіанти взаємодії користувача та оброблення даних метеостанції.

Застосунок розпочинається із запуску та ініціалізації основних компонентів. Якщо система знаходить збережені дані про попереднє з'єднання, вона автоматично відновлює сеанс. В головному меню користувач має можливість вибрати між переглядом інструкцій, даних метеостанції або виходом з програми.

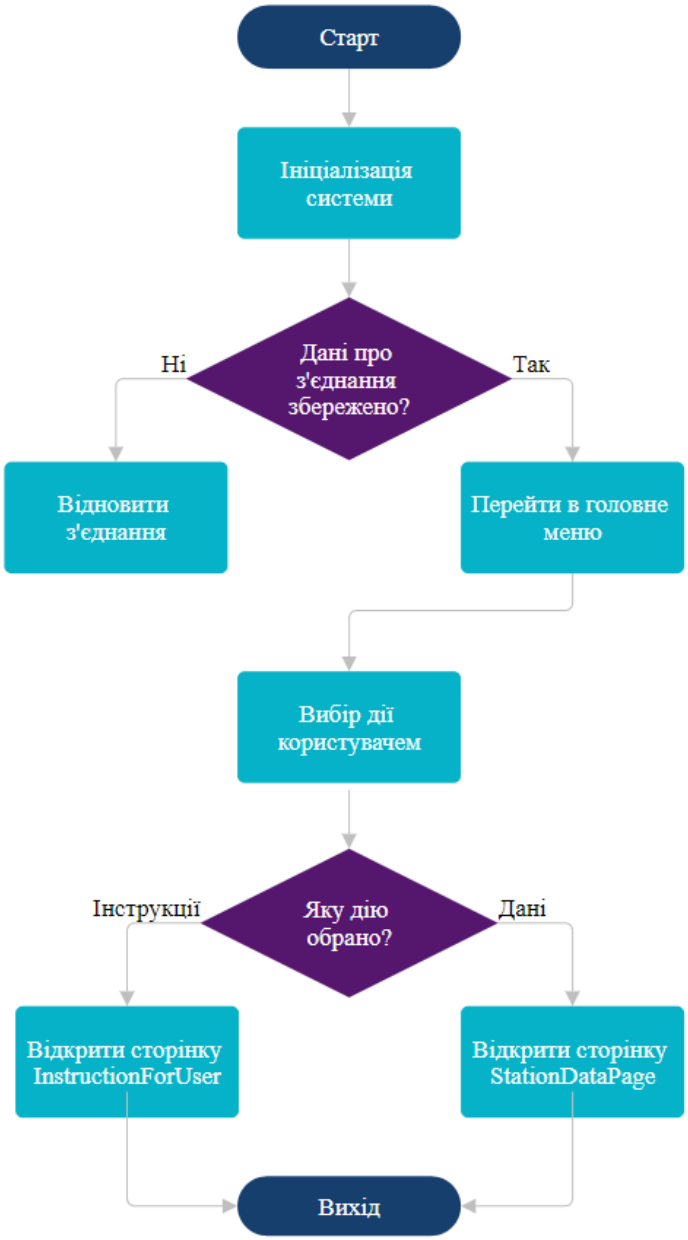

Рисунок 2.7 — Блок-схема алгоритму роботи застосунку

На сторінці з даними метеостанції інформація оновлюється в реальному часі, де користувач може спостерігати за показниками погоди та статусом датчиків. Система підтримує активне з'єднання з метеостанцією, регулярно оновлюючи дані та перевіряючи їх на предмет точності та валідності.

У разі вибору відключення, користувач може безпечно розірвати з'єднання з метеостанцією, після чого може повернутися до головного меню або завершити роботу з програмою.

Розробка мікрокомп’ютерної погодної станції та мобільного інтерфейсу, представляє собою комплексний підхід до інтеграції сучасних технологій у моніторинг погодних умов. Процес проєктування об'єднує теоретичні основи та практичні аспекти створення ефективної системи, яка починається з детального моделювання UML-діаграм, що дозволяє візуалізувати і оптимізувати архітектуру системи та взаємодії між її компонентами. Ключовими елементами системи є датчики, що вимірюють температуру, вологість, атмосферний тиск та інші критичні показники, які визначають погодні умови. Ці дані відображаються у мобільному застосунку, створюючи інтерфейс, доступний для користувача з можливістю оновлення даних в режимі реального часу.

Паралельно з технічною реалізацією, значна увага приділяється плануванню та розробці мобільного застосунку, який забезпечує інтерактивний доступ до даних станції. Розробка мобільного застосунку охоплює створення користувацького інтерфейсу, який не тільки відображає дані, але й дозволяє користувачам керувати налаштуваннями станції, переглядати історичні дані та отримувати повідомлення про погодні умови. Це вимагає глибокого розуміння потреб кінцевих користувачів та забезпечення надійності та точності передачі даних, що є ключовими для успішної експлуатації системи в різних погодних умовах. Завдяки комплексному підходу до проєктування та виконання, мікрокомп’ютерна погодна станція і мобільний застосунок виступають як взаємодоповнюючі компоненти однієї інтегрованої системи, здатної задовольняти сучасні вимоги до моніторингу погоди.

2.4 Характеристика об’єкту розробки

Об'єкт розробки представляє собою інноваційну метеостанцію, побудовану на основі потужного мінікомп'ютера Raspberry Pi 4 Model B, який слугує як центральний обчислювальний модуль. Основні вимоги до цієї системи включають:

— вимірювання температури та вологості повітря всередині приміщення, що дозволить користувачам отримувати точні дані про мікроклімат у їхньому житловому або робочому просторі;

— визначення атмосферного тиску та висоти над рівнем моря, що є важливим параметром для аналізу погодних умов та прогнозування погоди;

— відображення зібраних метеорологічних даних на Android пристрої, забезпечить наглядність та зручність спостереження за показниками в реальному часі;

— інтеграція Wi-Fi модуля для бездротової передачі даних, яка дозволить користувачам аналізувати погодні умови дистанційно через одну Wi-Fi мережу;

— гнучка архітектура системи та сумісність з різноманітним апаратним і програмним забезпеченням, що дозволить легко розширювати функціонал метеостанції шляхом додавання нових датчиків (наприклад, датчики UV-випромінювання, вимірювання якості повітря, анемометри для визначення швидкості та напрямку вітру) та впровадження нових функцій.

Цей проєкт метеостанції на базі Raspberry Pi 4 Model B може стати ідеальним рішенням не тільки для індивідуального використання вдома, але й слугувати цінним інструментом для навчальних закладів, наукових досліджень, а також для аграрного сектору, де вимоги до точності та актуальності погодних даних не надто високі. Ця система забезпечить відносно детальне відстеження погодних умов, враховуючи потребу в розширенні можливостей шляхом додавання нових датчиків або оновлення програмного забезпечення. Така гнучкість і масштабованість можуть перетворити цю метеостанцію на ресурс для будь-кого, хто прагне високої точності та ефективності в області погодного моніторингу.

2.5 Аналіз мікроконтролерних платформ та Raspberry Pi 4

На сьогоднішній день ринок пропонує широкий вибір домашніх метеостанцій, що входять до категорії IoT-проєктів. Аналізуючи цей сегмент, можна відзначити, що значна частина пристроїв заснована на платформах Arduino Uno та Raspberry Pi. Однак, існують і інші мікроконтролери та мікрокомп'ютери, які за розмірами є більш компактними та надають ширші функціональні можливості порівняно з вищезгаданими. Розробники постійно вдосконалюють

апаратні та програмні рішення, пропонуючи більш сучасні та потужні варіанти для реалізації IoT-проєктів, зокрема й домашніх метеостанцій [13].

Більшість моделей домашніх метеостанцій потребують з'єднання з сервером або хмарним сховищем для передачі зібраних даних, і це з'єднання, як правило, здійснюється за допомогою окремого Wi-Fi модуля. Така архітектура дозволяє не лише зберігати отримані дані в централізованому місці, але й забезпечує можливість віддаленого моніторингу та керування метеостанцією через Інтернет. Проте, використання додаткових модулів може ускладнювати процес налаштування та підвищувати вартість системи.

При визначенні оптимальної основи для проєкту, як мінікомп'ютера, основну увагу було приділено таким аспектам, як вартість, розміри, функціональні можливості, здатність до програмування у Raspberry Pi OS, а також наявність інтегрованого Wi-Fi модуля. В результаті ретельного аналізу, вибір був зроблений на користь Raspberry Pi 4, який оснащений вбудованим модулем Wi-Fi та потужним процесором, що забезпечує необхідну продуктивність для обробки даних метеостанції та взаємодії з хмарним сховищем.

Важливим етапом у процесі планування проєкту є ретельний порівняльний аналіз доступних на ринку додаткових модулів та аксесуарів для Raspberry Pi 4, а також альтернативних плат та мікроконтролерів від різних виробників. Такий аналіз допоможе визначити, які саме компоненти та рішення найкраще підійдуть для реалізації задуму проєкту, враховуючи їх технічні характеристики, сумісність, вартість та доступність. Вибір оптимальних апаратних та програмних складових є запорукою створення ефективної та надійної системи моніторингу погодних умов.

2.6 Типи датчиків у метеорологічних системах

На метеостанціях застосовуються різноманітні інструменти для моніторингу погодних умов, серед яких:

— термометри, що використовуються для фіксації вологості повітря в околицях станції, більшість метеостанцій оснащені двома термометрами, один

розташований на панелі управління для вимірювання температури всередині, а другий у вуличному модулі, що вимірює температуру навколишнього середовища;

— гігрометри відображають рівень вологості у повітрі, дозволяючи оцінити вміст водяної пари, вони забезпечують надійні дані про вміст водяної пари в атмосфері, що дозволяє ретельно оцінювати ступінь насиченості повітря парою;

— анемометри, призначені для вимірювання швидкості вітру, існують різні методи вимірювання — один із них використовує обертові чашки на відкритому просторі, інший – акустичні виміри вітру, що проходить через спеціальний отвір у приладі;

— датчики вологості є ключовими для багатьох сфер, включаючи споживчі, промислові, медичні, та дозволяють точно моніторити рівень вологості в атмосфері;

— анемометри також використовуються для визначення напрямку вітру, з використанням тих самих принципів, що застосовуються для вимірювання його швидкості;

— барометри, що мають важливе значення для прогнозування погодних умов, вимірюють атмосферний тиск, зміна тиску часто вказує на майбутні зміни погоди;

— датчики тиску використовуються для вимірювання атмосферного тиску, а також у якості висотомірів та барометрів;

— дощоміри фіксують обсяг опадів, використовуючи різні технології для збору та аналізу води;

— датчики ультрафіолетового та сонячного випромінювання допомагають оцінити рівень ультрафіолетової радіації та загальне сонячне випромінювання;

— детектори блискавки дозволяють виявляти електричні розряди на значній відстані, забезпечуючи цінну інформацію про наближення грозових фронтів;

— датчики вологості листя та ґрунту корисні для аграріїв та садівників,

— оскільки вони допомагають контролювати стан росту рослин та ефективно планувати зрошення, вимірюючи вологість навколишнього середовища та безпосередньо в ґрунті.

Ці датчики забезпечують точні дані для оптимізації водного балансу та здоров'я рослин, підвищуючи урожайність та якість сільськогосподарських культур.

У контексті зростаючої стурбованості щодо зміни клімату та екологічних умов, деякі виробники метеостанцій інтегрували датчики якості повітря до своїх моделей. Ці датчики вимірюють рівень забруднення повітря, включаючи концентрацію твердих частинок, що надає користувачам цінну інформацію про екологічні умови в їхньому середовищі. Така інформація може бути особливо корисною для людей з респіраторними захворюваннями та для всіх, хто прагне зберегти здоров'я в умовах міського середовища.

Саме тому, сучасні метеостанції використовують широкий спектр датчиків та приладів для забезпечення комплексного моніторингу погодних умов та навколишнього середовища. Це не тільки сприяє покращенню точності погодних прогнозів, але й надає важливу інформацію для наукових досліджень, сільськогосподарської діяльності, екологічного моніторингу, а також для особистого використання і забезпечення комфорту та безпеки.

2.6.1 Термічні вимірювальні модулі

Наразі наявні численні датчики та модулі, призначені для фіксації температури та інших критичних індикаторів, що впливають на комфортне існування людини, а також на стан різних предметів і живих організмів. Ці компоненти можуть бути втілені в прості метеостанції, системи клімат-контролю, а також у «розумні» будинки для забезпечення необхідних умов у житлових просторах, на виробництві та в інших сценаріях.

Сенсори із родини DHT відзначаються особливою популярністю серед користувачів Raspberry Pi, завдяки їх простоті інтеграції та програмування, а також

доступній ціновій політиці. У лінійці DHT виокремлюють три основні типи датчиків: DHT11, DHT22 та DHT21, кожен з яких має свої унікальні характеристики.

Перші два датчики, DHT11 і DHT22, зовні дуже схожі і мають аналогічні методи підключення, як показано на рисунку 2.8 Відрізняється DHT21, який обладнаний захисним корпусом. Це робить його придатним для зовнішнього використання, адже корпус надійно захищає датчик від впливу пилу, бруду та опадів.

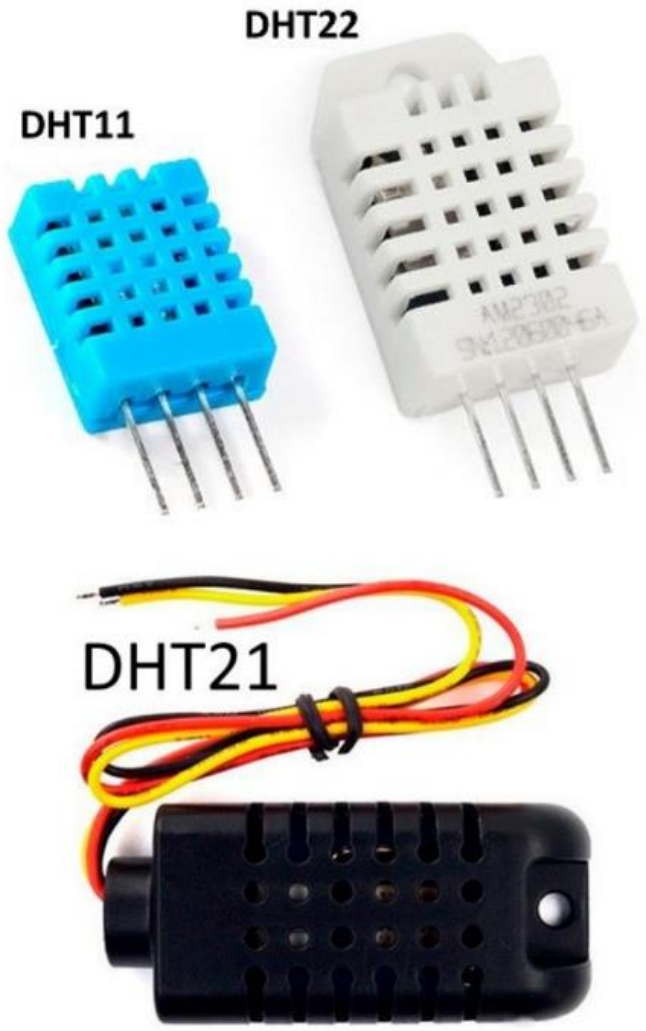

Рисунок 2.8 — Датчики температури DHT11, DHT22 та DHT21

Оцінюючи модулі за ключовими характеристиками, такими як точність вимірювань, діапазон вимірювання та вартість, ми маємо наступне:

— DHT11 визначає вологість у межах від 20 до 80% з похибкою ±5% і температуру від 0°C до +50°C з похибкою ±2°C, проводячи виміри кожну секунду;

— DHT22 забезпечує вимірювання вологості від 0 до 100% з точністю ±2% та температури в діапазоні від мінус 40°C до +125°C з точністю ±0.5°C, опитуючись кожні дві секунди;

— DHT21 також має вологість в діапазоні від 0 до 100% з точністю ±2% та температуру від мінус 40°C до +80°C з точністю ±0.5°.

При порівнянні вартості модулів виділяється DHT11 з його вигідною ціною, що обумовлено великим попитом і простотою конструкції. В той же час, DHT21 і DHT22 мають вищу ціну через кращу точність вимірювань, ширший діапазон температур, а у випадку DHT22 — також має захисний корпус, який забезпечує додатковий захист від зовнішніх впливів [14, 15].

Виходячи з цього аналізу, DHT22 видається найкращим варіантом для використання у домашніх метеостанціях. Його переваги, такі як оптимальна ціна, компактність, надійність і простота у використанні, роблять його ідеальним вибором для тих, хто не прагне вимірювати досить високі чи низькі температури, але хоче забезпечити точність і стабільність даних протягом тривалого часу.

2.6.2 Барометричні вимірювальні модулі

Серед найбільш вживаних і доступних датчиків атмосферного тиску виділяються продукти компанії BOSH: BMP085, BMP180 та GY-BMP280-3.3, серед яких останній представляє собою удосконалену версію з покращеними характеристиками.

Датчик GY-BMP280-3.3 (див. рисунок 2.9) було розроблено з урахуванням потреб у компактних розмірах і ефективному споживанні енергії. Це робить його ідеальним вибором для застосувань, таких як навігаційні системи, створення погодних прогнозів, таких як температура, атмосферний тиск та висота [16].

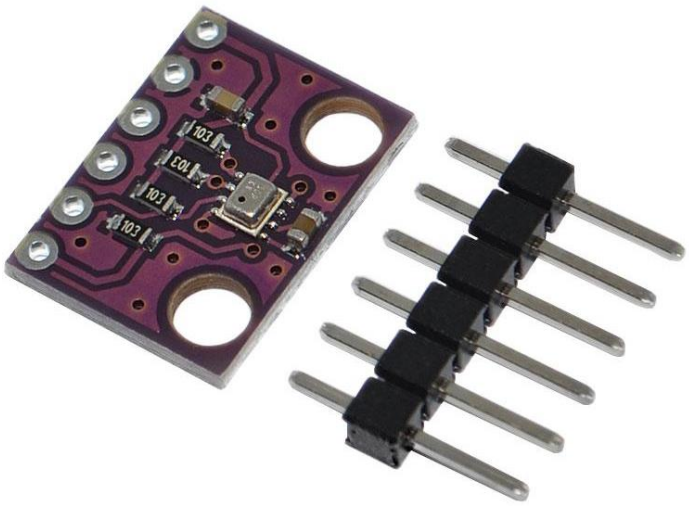

Рисунок 2.9 — Датчик атмосферного тиску GY-BMP280-3.3

Датчик GY-BMP280-3.3 має наступні технічні характеристики:

— Мікросхема BMP280;

— напруга живлення від 1,7В до 3,6 В;

— швидкість інтерфейсу I2C до 3,4 МГц;

— максимальне значення тиску 1100 гПа;

— рівень шуму 0,2 Па;

— чутливість по висоті 1 м;

— дозвіл АЦП 20 біт;

— підтримка роботи в трьох режимах (режим низького енергоспоживання або режим «сну», вимірювання необхідних параметрів за командою та самостійне вимірювання параметрів через заданий час).

Атмосферний тиск, який вимірює датчик BMP085 (див. рисунок 2.10), включає наступні технічні параметри [17]:

— діапазон вимірювань тиску від 300 до1100 ГПа, що відповідає висоті від 9000 метрів над рівнем моря до 500 метрів;

— напруга живлення знаходиться в межах від 1,8В до 3,6 В;

— споживаний струм 5 мкА при частоті опитування один раз на секунду;

— точність вимірювання від 0.5 метра в найшвидшому режимі та 0.25 метра в найбільш точному;

— час вимірювання 3 мс.;

— оснащений інтегрованим термометром для вимірювання температури.

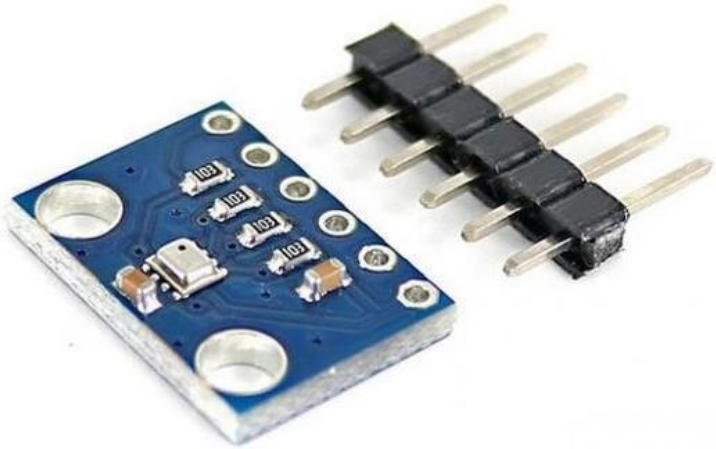

Рисунок 2.10 — Датчик атмосферного тиску BMP085

Датчик BMP180 (див. рисунок 2.11) є доступним і легким у використанні сенсором, призначеним для вимірювання атмосферного тиску та температури. Цей датчик часто використовується для визначення висоти над рівнем моря та застосовується у метеорологічних станціях. Він складається з п'єзорезистивного датчика тиску, термодатчика, аналого-цифрового перетворювача, власної пам'яті, оперативної пам'яті та мікроконтролера [18].

Рисунок 2.11 — Датчик атмосферного тиску BMP180

Основні технічні характеристики:

— діапазон вимірювань тиску від 225 до 825 мм ртутного стовпа;

— напруга живлення між 3.3В та 5В;

— споживаний струм 0.5 мА;

— підтримка інтерфейсу I2C для комунікації;

— час реакції 4.5 мс;

— габарити датчика (15x14) мм.

У рамках розробки нашого проєкту обрано датчик атмосферного тиску моделі GY-BMP280-3.3.

2.6.3 Технічні характеристики мінікомп’ютера Raspberry Pi 4

Модель Raspberry Pi 4 Model B (див. рисунок 2.12) принесла ряд значних покращень, включаючи збільшену швидкість процесора, вдосконалення в обробці мультимедійного контенту, розширення обсягу та швидкості доступу до оперативної пам'яті, а також оновлені можливості мережевого з'єднання у порівнянні з попередником, Raspberry PI3 Model B+. При цьому збережено ефективність енергоспоживання та сумісність з різноманітними аксесуарами. Raspberry Pi 4 Model B надає користувачам потужність стаціонарних комп'ютерів початкового рівня на основі X86 архітектури [19].

Рисунок 2.12 — Мінікомп’ютер Raspberry Pi 4 Model B

Серед вирішальних переваг пристрою варто виділити чотириядерний процесор нового покоління з 64-бітною архітектурою для високої продуктивності, можливість підключення двох дисплеїв із роздільною здатністю до 4k через micro-HDMI, апаратне декодування 4K відео на швидкості 60 кадрів за секунду, від 1 до 4 Гб оперативної пам'яті, підтримка Wi-Fi на 2,4 та 5,0 ГГц, Bluetooth 5.0, Gigabit Ethernet, два USB 3.0 порти та можливість розширення за допомогою PoE через додатковий модуль PoE HAT.

Технічні характеристики включають:

— процесор Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz;

— оперативна пам'ять 1GB LPDDR4-2400 SDRAM;

— живлення 5V 3A;

— бездротові можливості Bluetooth та Wi-Fi;

— кількість ядер процесора 4;

— GPIO контактів 40 pins;

— частота процесора 1.5 ГГц;

— інтерфейси 3.5 Jack, RJ45, USB, USB Type-C і microHDMI;

— мережеві можливості Ethernet 10/100/1000 і Wi-Fi 802.11ac;

— сховище eMMC 16 GB eMMC;

— GPU VideoCore VI;

— USB порти 4 шт..

Для ефективної роботи з мінікомп'ютером Raspberry Pi 4 Model B було інтегровано підтримку операційної системи Raspberry Pi OS. Ця система забезпечує широкі можливості для розробки та реалізації різноманітних проєктів, сумісна з більшістю бібліотек, розроблених для попередніх версій Raspberry Pi. Відмінності Raspberry Pi 4 Model B полягають не тільки в підвищеній продуктивності та збільшеній кількості оперативної пам'яті, але й у наявності удосконаленого тепловідведення, що дозволяє підтримувати оптимальну роботу процесора навіть при високих навантаженнях. Raspberry Pi 4 обладнаний декількома цифровими

GPIO пінами, що дозволяє використовувати широкий спектр зовнішніх модулів та датчиків для реалізації проєктів різної складності.

Розпінування мінікомп’ютера Raspberry Pi 4 Model B представлено на рисунку 2.13. На платі розташована гребінка з 40 контактів, які умовно можна поділити на піни живлення та піни вводу-виводу. За замовчуванням будемо використовувати саме нумерацію фізичних контактів.

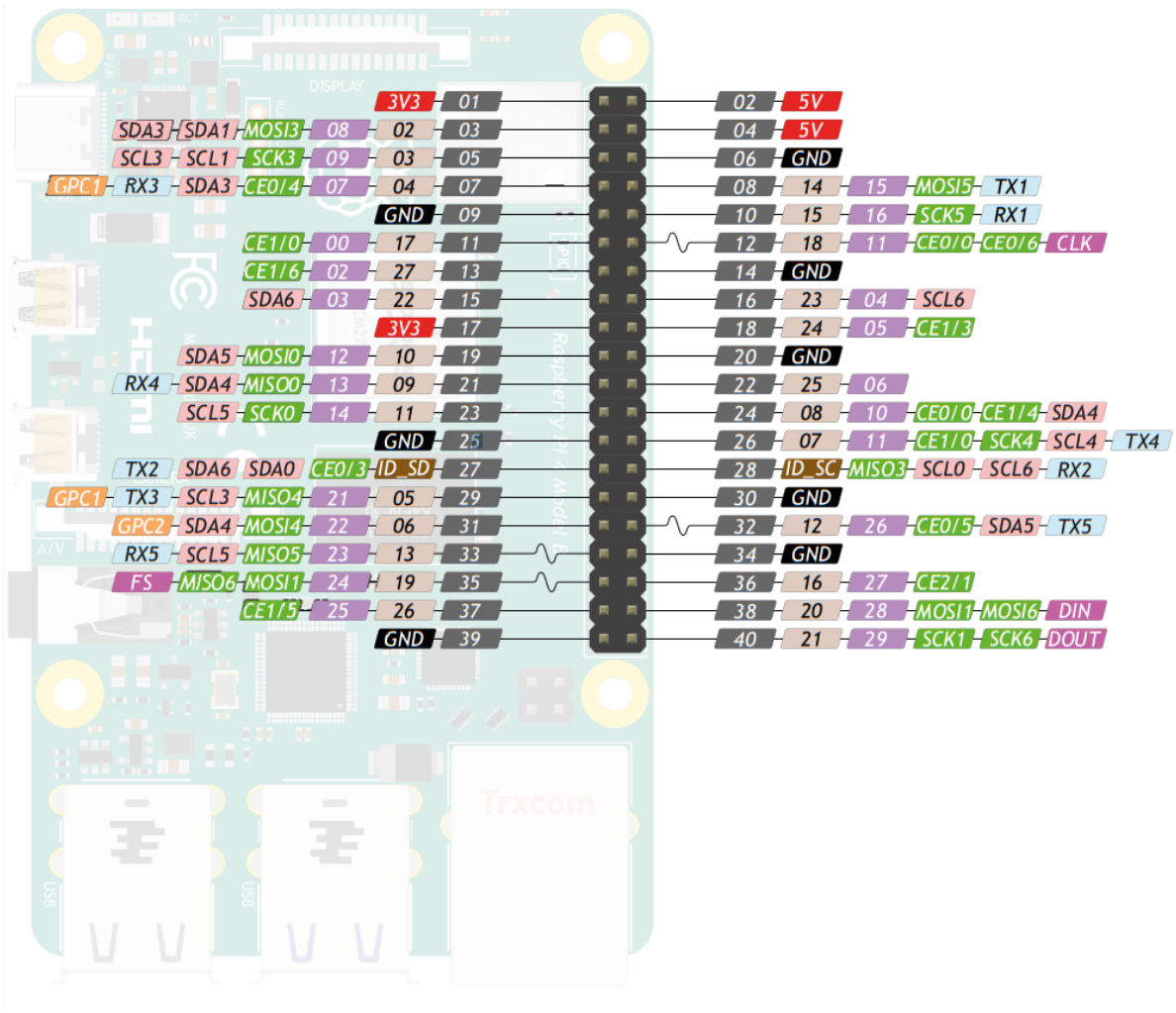

Рисунок 2.13 — Розпінування плати Raspberry Pi 4 Model B

Піни живлення:

— 5V 2 піна, 2 та 4, вхідні піни для підключення зовнішнього джерела напругою 5 вольт, на платі контакти електрично пов'язані між собою;

— 3V3 2 піна, 1 та 17, вихідні піни від стабілізатора напруги із виходом 3.3 вольта, на платі контакти електрично пов'язані між собою;

— GND 8 пінів, 6, 9, 14, 20, 25, 30, 34 та 39, виведення спільної землі, на платі контакти електрично пов'язані між собою.

Піни вводу-виводу (GPIO) — максимальна напруга, яка може витримати піни введення-виводу становить 3.3 В, а не 5 В.

Піни вводу-виводу (GPIO) —максимальна напруга, яка може витримати піни введення-виводу становить 3.3 В, а не 5 В.

Піни загального призначення — 28 пінів, 3,5, 7,8,10-13, 15, 16, 18, 19, 21-24, 26-29, 31-33, 35-38 та 40. Можуть працювати як на вхід, так і вихід. Логічний рівень одиниці — 3.3 В, а нуля — 0 В.

BCM / GPIO Pin —28 пінів, нумерація контактів процесора Broadcom, може стати в нагоді при роботі з пакетом Rpi.GPIO.

WiringPi Pin — 28 пінів, нумерація контактів для пакета Wiring Pi, це Arduino-подібна бібліотека для роботи з GPIO-контактами.

ШІМ — дозволяє виводити аналогову напругу у вигляді ШІМ-сигналу із цифрових значень:

— PWM0 піни 12 и 18;

— PWM1 піни 13 и 19.

I²C Для спілкування Raspberry Pi з платами розширення та сенсорами за інтерфейсом I²C:

— I²C0 піни SDA0/27 та SCL0/28. Контакти зарезервовані для завантаження на Raspberry Pi;

— I²C1 піни SDA1/3 та SCL1/5;

— I²C3 піни SDA3/3/7 та SCL3/5/29;

— I²C4 піни SDA4/24/31 та SCL4/21/26;

— I²C5 піни SDA5/19/32 та SCL5/23/33;

— I²C6 піни SDA6/15 та SCL5/16.

SPI Для спілкування Raspberry Pi з платами розширення та сенсорами за інтерфейсом SPI:

— SPI0 піни MOSI/19, MISO/21, SCK/23, CE0/24 та CE1/26;

— SPI1 піни MOSI/28, MISO/35, SCK/40, CE0/12, CE1/11 та CE2/36;

— SPI3 піни MOSI/3, MISO/28, SCK/5, CE0/27 та CE1/18;

— SPI4 піни MOSI/31, MISO/29, SCK/26, CE0/7 та CE1/22;

— SPI5 піни MOSI/8, MISO/33, SCK/10, CE0/32 та CE1/37;

— SPI6 піни MOSI/38, MISO/35, SCK/40, CE0/12 та CE1/13.

UART Для спілкування Raspberry Pi з платами розширення та сенсорами за інтерфейсом UART:

— UART1 піни TX/8 та RX/10;

— UART2 піни TX/27 та RX/28;

— UART3 піни TX/29 та RX/7;

— UART4 піни TX/24 та RX/21;

— UART5 піни TX/32 та RX/33.

3 Програмна реалізація метеостанції та розробка android-застосунку

3.1 Інтеграція та налаштування датчика DHT22

Одним з ключових компонентів розробленої мікрокомп'ютерної метеостанції є датчик DHT22 — високоточний цифровий сенсор для вимірювання температури та відносної вологості повітря. Інтеграція цього датчика з мікрокомп'ютером Raspberry Pi 4 Model B є важливим етапом у процесі створення функціональної системи моніторингу погодних умов.

Перш за все, необхідно правильно підключити датчик DHT22 до відповідних контактів (пінів) на платі Raspberry Pi (рис 3.1). Згідно з технічною документацією, підключення здійснюється наступним чином:

— контакт VCC датчика підключається до виводу 3.3V на Raspberry Pi 4 Model B (пін 1), що забезпечує живлення сенсора;

— контакт DIO (цифровий вхід/вихід) датчика підключається до GPIO (General Purpose Input/Output) на Raspberry Pi 4 Model B (пін 7), що дозволяє передавати та отримувати дані;

— контакт GND (заземлення) датчика підключається до заземлюючого виводу на Raspberry Pi 4 Model B (пін 6).

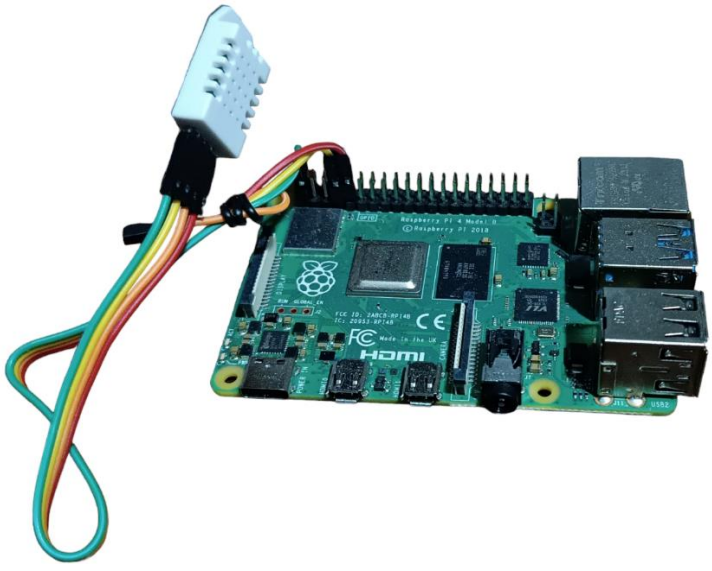

Рисунок 3.1 — Результат підключення модуля DHT22

Після фізичного підключення датчика необхідно налаштувати програмне забезпечення для взаємодії з ним та отримання даних про температуру та вологість. У цьому проєкті використовуються наступні бібліотеки для мови програмування Python:

— pigpio бібліотека надає програмний інтерфейс для керування контактами GPIO (General Purpose Input/Output) на платформі Raspberry Pi, вона дозволяє встановлювати режими роботи окремих пінів (вхід/вихід, цифровий/аналоговий), відправляти та отримувати сигнали, налаштовувати таймери та обробники подій, бібліотека pigpio забезпечує високу продуктивність та точність при роботі з GPIO, що є критично важливим для підключення різноманітних сенсорів та датчиків, таких як DHT22;

— pigpio_dht бібліотека розширює функціональність pigpio, надаючи спеціалізовані засоби для взаємодії з цифровими датчиками температури та вологості серії DHT (DHT11, DHT22 тощо), вона реалізує необхідні протоколи зв'язку та алгоритми зчитування даних від цих датчиків, що значно спрощує процес їх інтеграції в проєкти на базі Raspberry Pi, бібліотека pigpio_dht дозволяє зручно отримувати значення температури та вологості у форматі, зрозумілому для подальшої обробки;

— time стандартна бібліотека мови програмування Python, яка надає функції для роботи з часом та затримками, у контексті цього проєкту вона використовується для реалізації затримки між послідовними зчитуваннями даних з датчика DHT22,це необхідно для того, щоб давати датчику достатньо часу для стабілізації показників після кожного зчитування.

Використання цих бібліотек забезпечує ефективне та зручне підключення датчика DHT22 до Raspberry Pi 4 Model B, а також дозволяє отримувати точні дані про температуру та вологість повітря в потрібному форматі. Комбінація pigpio, pigpio_dht та time створює надійну основу для реалізації функції моніторингу погодних умов у складі мікрокомп'ютерної метеостанції [20].

Для зчитування даних з датчика DHT22 було реалізовано скрипт на Python,

який включає наступний код:

```
GPIO_PIN = 4 # Вказуємо номер GPIO-піну, до якого підключений датчик
dht_sensor = pigpio_dht.DHT22(GPIO_PIN) # Створюємо об'єкт датчика
pi = pigpio.pi() # Ініціалізуємо бібліотеку pigpio
if not pi.connected:
    exit() # Перевіряємо підключення до pigpio
try:
    while True:
        result = dht_sensor.read() # Зчитуємо дані з датчика
        print(f"Temperature: {result['temp_c']} °C, Humidity:
{result['humidity']}%") # Виводимо результат
        time.sleep(2) # Затримка в 2 секунди перед наступним
зчитуванням
finally:
    pi.stop() # Зупиняємо роботу бібліотеки pigpio
```

Скрипт виконує наступні дії:

— встановлює номер GPIO-піну, до якого підключений датчик DHT22;

— створює об'єкт датчика та ініціалізує бібліотеку pigpio;

— перевіряє успішність підключення до pigpio;

— в циклі зчитує дані з датчика та виводить поточні значення температури (в градусах Цельсія) та відносної вологості (у відсотках);

— робить затримку в 2 секунди перед наступним зчитуванням даних;

— після виходу з циклу зупиняє роботу бібліотеки pigpio.

Таким чином, завдяки правильному підключенню датчика DHT22 до плати Raspberry Pi 4 Model B та реалізації відповідного програмного забезпечення, мікрокомп'ютерна метеостанція отримує можливість точного вимірювання температури та вологості повітря, що є критично важливим для моніторингу погодних умов та подальшого аналізу зібраних даних.

Демонстрацію роботи скрипту в командному рядку Raspberry Pi 4 Model B можемо побачити на рисунку 3.2.

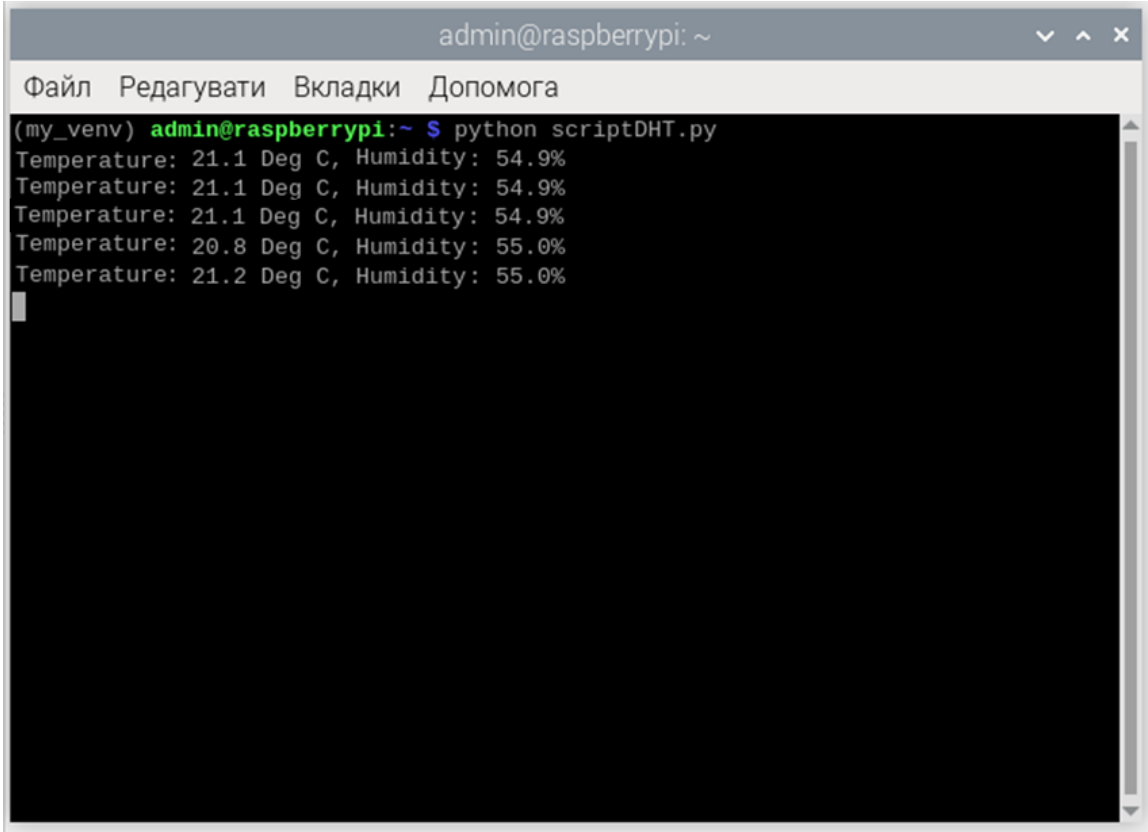

Рисунок 3.2 — Результати роботи скрипту для зчитування датчика DHT22

На наведеному скриншоті ми бачимо результат роботи програмного скрипту scriptDHT.py на Raspberry Pi. Скрипт періодично зчитує показники температури та відносної вологості з підключеного датчика DHT22 та виводить їх у термінал.

На виводі ми спостерігаємо рядків з даними, де для кожного виміру вказується температура у градусах Цельсія та відносна вологість у відсотках. Наприклад, один з рядків показує значення «Temperature: 21.2 Deg C, Humidity: 55.0%».

Загалом, скриншот демонструє успішну інтеграцію датчика DHT22 з Raspberry Pi та коректну роботу програмного забезпечення для зчитування та відображення даних про температуру та вологість повітря в термінальному вікні.

3.2 Інтеграція та налаштування датчика GY-BMP280-3.3

Крім датчика DHT22 для вимірювання температури та вологості, у складі мікрокомп'ютерної метеостанції також використовується датчик GY-BMP280-3.3 для визначення атмосферного тиску та розрахунку висоти над рівнем моря. Цей

високоточний цифровий датчик забезпечує надійне отримання даних про атмосферні умови, що є важливим для повноцінного моніторингу погодних явищ.

Інтеграція датчика GY-BMP280-3.3 з мікрокомп'ютером Raspberry Pi 4 Model B (рис. 3.3) вимагає ретельного підключення до відповідних контактів згідно з технічною документацією:

— контакт VCC датчика підключається до виводу 3.3V на Raspberry Pi 4 Model B (пін 17) для живлення сенсора;

— контакт GND (заземлення) датчика підключається до заземлюючого виводу Ground (GND) на Raspberry Pi 4 Model B (пін 25);

— контакт SCL датчика підключається до SCL-піну I2C інтерфейсу на Raspberry Pi 4 Model B (пін 5, GPIO 3) для передачі тактового сигналу;

— контакт SDA датчика підключається до SDA-піну I2C інтерфейсу на Raspberry Pi 4 Model B (пін 3, GPIO 2) для передачі даних.

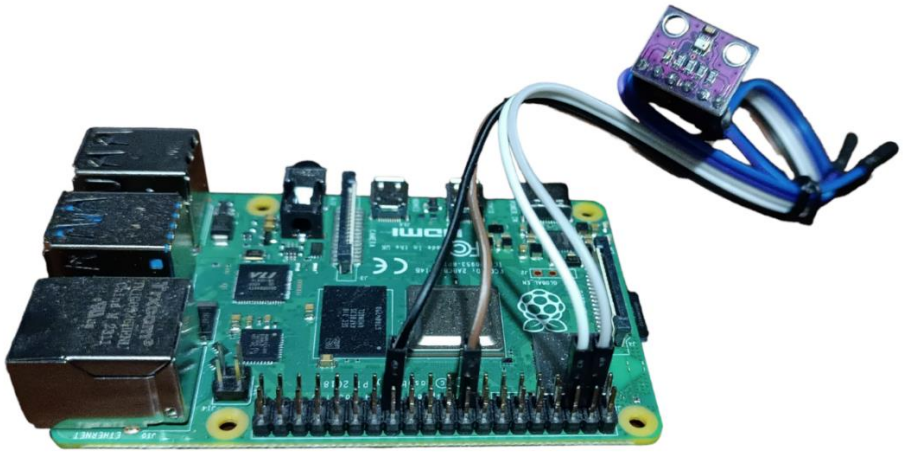

Рисунок 3.3 — Результат підключення модуля GY-BMP280-3.3

Після фізичного підключення датчика необхідно налаштувати програмне забезпечення для взаємодії з ним та отримання даних про атмосферний тиск та висоту. У цьому проєкті використовуються наступні бібліотеки для мови програмування Python:

— smbus2 бібліотека надає програмний інтерфейс для роботи з протоколом I2C (Inter-Integrated Circuit) на платформі Raspberry Pi, протокол I2C є

серійним інтерфейсом передачі даних, який широко використовується для підключення різноманітних периферійних пристроїв, таких як датчики, дисплеї, мікроконтролери та інші компоненти, а бібліотека smbus2 дозволяє відкривати та закривати канали зв'язку I2C, виконувати читання та запис даних, а також керувати адресами пристроїв на шині I2C;

— bmp280 бібліотека спеціально розроблена для роботи з цифровим датчиком атмосферного тиску BMP280 від виробника Bosch Sensortec, вона інкапсулює всю логіку взаємодії з датчиком через інтерфейс I2C, надаючи високорівневий програмний інтерфейс для зчитування показників атмосферного тиску, а також розрахунку висоти над рівнем моря на основі виміряного тиску, а бібліотека bmp280 абстрагує складні деталі роботи з датчиком, такі як ініціалізація, калібрування, обробка сирих даних тощо, дозволяючи отримувати точні та калібровані значення тиску та висоти у зручному форматі.

Комбінація бібліотек smbus2, bmp280 та time забезпечує ефективну інтеграцію датчика GY-BMP280-3.3 з мікрокомп'ютером Raspberry Pi 4 Model B, дозволяючи отримувати точні дані про атмосферний тиск та висоту над рівнем моря, які є критично важливими для комплексного аналізу та моніторингу погодних умов [21].

Для зчитування даних з датчика GY-BMP280-3.3 було реалізовано скрипт на Python, який включає наступний код:

```
I2C_PORT = 1 # Вказуємо номер I2C-порту на Raspberry Pi
BMP280_ADDRESS = 0x76 # Задаємо адресу датчика BMP280
bus = smbus2.SMBus(I2C_PORT) # Ініціалізуємо шину I2C
sensor = bmp280.BMP280(i2c_dev=bus, i2c_addr=BMP280_ADDRESS) # Створюємо об'єкт датчика
while True:
    pressure = sensor.get_pressure() # Зчитуємо атмосферний тиск
    altitude = sensor.get_altitude() # Зчитуємо висоту
    print(f"Pressure: {pressure:.2f} hPa, Altitude: {altitude:.2f} m") # Виводимо результат
    time.sleep(2) # Затримка в 2 секунди до зчитування
```

Скрипт виконує наступні дії:

— встановлює номер I2C-порту та адресу датчика BMP280;

— ініціалізує шину I2C та створює об'єкт датчика;

— в циклі зчитує поточні значення атмосферного тиску (в гектопаскалях) та висоти (в метрах) з датчика;

— виводить отримані дані у термінал;

— робить затримку в 2 секунди перед наступним зчитуванням.

Використання бібліотек smbus2 та bmp280 забезпечує коректну взаємодію з датчиком GY-BMP280-3.3 через інтерфейс I2C, дозволяючи отримувати точні виміри атмосферного тиску та висоти над рівнем моря. Ці дані, разом з показниками температури та вологості від датчика DHT22, створюють комплексну картину погодних умов, необхідну для ефективного моніторингу та аналізу метеорологічної ситуації.

Демонстрацію роботи скрипту в командному рядку Raspberry Pi 4 Model B можемо побачити на рисунку 3.4.

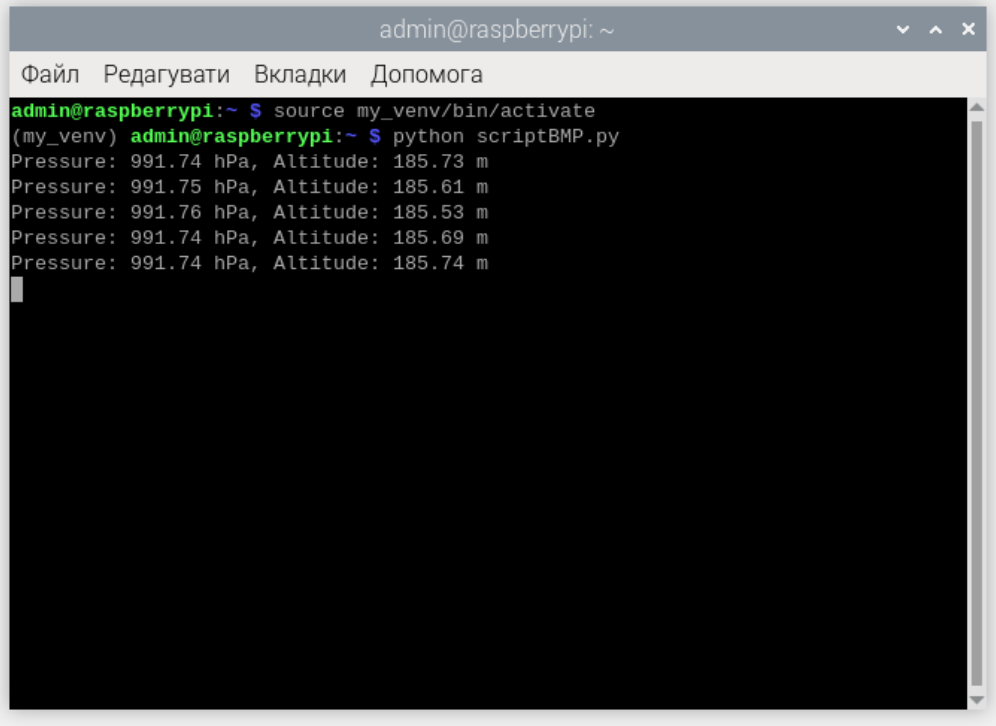

Рисунок 3.4 — Результати роботи скрипту для зчитування датчика GY-BMP280-3.3

На наведеному скриншоті ми бачимо результат роботи програмного скрипту scriptBMP.py на Raspberry Pi. Скрипт періодично зчитує показники атмосферного тиску та висоти з підключеного датчика BMP і виводить їх у термінал.

На виводі ми спостерігаємо кілька рядків з даними, де для кожного виміру вказується тиск у гектопаскалях (hPa) та висота у метрах (m). Наприклад, один з рядків показує значення «Pressure: 991.74 hPa, Altitude: 185.73 m».

Скриншот виконуваного скрипта демонструє успішну інтеграцію датчика BMP з Raspberry Pi та коректну роботу програмного забезпечення для зчитування та відображення даних про атмосферний тиск і висоту в термінальному вікні.

3.3 Інтеграція системи метеостанції з базою даних для зберігання даних

У попередніх підрозділах було детально описано підключення датчиків температури та вологості DHT22 та процес підключення датчика атмосферного тиску і висоти GY-BMP280-3.3 мінікомп’ютера Raspberry Pi 4 Model B.

Підключимо обидва датчика до зазначених пінів з попередніх підрозділів до мінікомп’ютера Raspberry Pi 4 Model B (рис. 3.5).

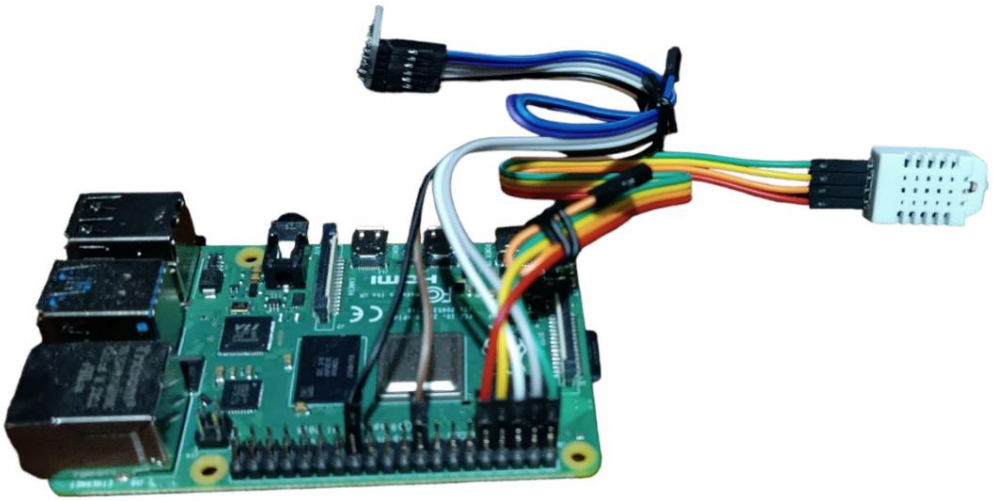

Рисунок 3.5 — Підключення датчиків DHT22 і GY-BMP280-3.3

Для зберігання даних, зчитаних з датчиків, ми використаємо реляційну базу даних MariaDB. Ця база даних забезпечить надійне зберігання та легкий доступ до

даних для подальшого аналізу та візуалізації. Використовуючи MariaDB, ми зможемо створити таблиці для зберігання інформації про температуру, вологість та атмосферний тиск, а також реалізувати запити для отримання необхідних даних у зручному вигляді [22, 23].

Встановлення та налаштування СУБД MariaDB здійснюємо через термінал, виконуючи наступні команди:

— оновлення списку пакетів sudo apt update

— встановлення MariaDB sudo apt install mariadb-server

— налаштування безпеки MariaDB sudo mysql_secure_installation

Ця команда з налаштування безпеки дозволяє встановити пароль для кореневого користувача (root), видалити анонімних користувачів, заборонити підключення root ззовні та видалити тестову базу даних.

Після встановлення та налаштування MariaDB було створено нову базу даних «sensors» та таблицю «sensor_data» для зберігання даних з датчиків. Виконані команди для створення таблиці SQL:

```
CREATE DATABASE sensors;
USE sensors;
CREATE TABLE sensor_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    temperature DECIMAL(5,2),
    humidity DECIMAL(5,2),
    pressure DECIMAL(7,2),
    altitude DECIMAL(7,2),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    dht22_status VARCHAR(50),
    bmp280_status VARCHAR(50)
);
```

Опис та призначення полів таблиці «sensor_data»:

— id унікальний ідентифікатор для кожного запису, використовується для легкого пошуку та індексації даних;

— temperature зберігає значення температури від датчика DHT22 з точністю до двох знаків після коми;

— humidity зберігає значення вологості від датчика DHT22 з точністю до двох знаків після коми;

— pressure зберігає значення атмосферного тиску від датчика GY-BMP280-3.3 з точністю до двох знаків після коми;

— altitude зберігає висоту над рівнем моря, розраховану на основі даних з датчика GY-BMP280-3.3, з точністю до двох знаків після коми;

— timestamp зберігає час останнього оновлення даних, використовується для відстеження часу зчитування даних;

— dht22_status зберігає стан датчика DHT22 («OK» або «FAIL»);

— bmp280_status зберігає стан датчика GY-BMP280-3.3 («OK» або «FAIL»).

Для зчитування даних з датчиків та їх збереження у базі даних ми створили скрипт на мові Python. Цей скрипт поєднує логіку попередніх скриптів для роботи з датчиками DHT22 та GY-BMP280-3.3, а також додає функціональність для підключення до бази даних MariaDB та запису зчитаних даних у таблицю «sensor_data». Використання MariaDB забезпечить надійне зберігання та легкий доступ до даних для подальшого аналізу та візуалізації. Таким чином, ми зможемо ефективно керувати великими обсягами даних та забезпечити їх цілісність і доступність.

Для реалізації скрипта було імпортовано наступні бібліотеки:

— smbus2 та bmp280 для роботи з датчиком GY-BMP280-3.3;

— pigpio та pigpio_dht для роботи з датчиком DHT22;

— time для роботи з часом;

— mysql.connector для підключення до бази даних MariaDB;

— logging для реєстрації помилок та інформаційних повідомлень.

Ініціалізація датчиків:

— створюється екземпляр класу BMP280 для роботи з датчиком атмосферного тиску;

— створюється екземпляр класу DHT22 для роботи з датчиком температури та вологості;

— ініціалізується з'єднання з GPIO Raspberry Pi за допомогою бібліотеки pigpio.

```
I2C_PORT = 1
BMP280_ADDRESS = 0x76
bus = smbus2.SMBus(I2C_PORT)
sensor = bmp280.BMP280(i2c_dev=bus, i2c_addr=BMP280_ADDRESS

GPIO_PIN = 4
dht_sensor = pigpio_dht.DHT22(GPIO_PIN)
pi = pigpio.pi()
```

Для конфігурації підключення до БД задаються параметри для підключення до бази даних MariaDB, такі як ім'я користувача, пароль, хост та назва бази даних:

```
db_config = {
  'user': 'root',
  'password': 'rtpass',
  'host': 'localhost',
  'database': 'sensors',
  'raise_on_warnings': True
}
```

В основному нескінченному циклі, виконуються наступні дії.

Зчитуються дані з датчика GY-BMP280-3.3 (атмосферний тиск та висота над рівнем моря).

```
try:
    pressure = round(sensor.get_pressure(), 2)  # Округлення до
двох знаків після коми
    altitude = round(sensor.get_altitude(), 2)  # Округлення до
двох знаків після коми
```

```
        bmp280_status = "OK"
            logging.info("Info Save")
        except Exception as e:
            print(f"Помилка зчитування з GY-BMP280-3.3: {e}")
            bmp280_status = "FAIL"
            pressure, altitude = None, None  # Встановлюємо значення як
        None, якщо зчитування не вдалося
            logging.error("Error")
```

Зчитуються дані з датчика DHT22 (температура та вологість).

```
        result = dht_sensor.read()
        dht22_status = "OK" if result['valid'] else "FAIL"
```

Визначається статус роботи кожного датчика («OK» або «FAIL»).

```
        bmp280_status = "OK" if pressure is not None else "FAIL"
        dht22_status = "OK" if result['valid'] else "FAIL"
```

Дані разом зі статусом датчиків вставляються у таблицю «sensor_data» бази даних «sensors».

```
        insert_query = ("INSERT INTO sensor_data (temperature, humidity,
        pressure, altitude, dht22_status, bmp280_status) "
                        "VALUES (%s, %s, %s, %s, %s, %s)")
        data_tuple = (result['temp_c'] if result['valid'] else None,
                      result['humidity'] if result['valid'] else None,
                      pressure, altitude, dht22_status, bmp280_status)
        cursor.execute(insert_query, data_tuple)
        db_connection.commit()
```

Перевіряється кількість записів у таблиці «sensor_data». Якщо кількість перевищує 1000 записів, таблиця очищується для економії пам’яті.

```
        cursor.execute("SELECT COUNT(*) FROM sensor_data")
        count = cursor.fetchone()[0]
        if count >= 1000:
```

```
            cursor.execute("TRUNCATE TABLE sensor_data")
                db_connection.commit()
                logging.info("Table truncated due to reaching 1000 records.")
```

Цей скрипт дозволяє нам зчитувати дані з датчиків DHT22 та GY-BMP280-3.3, зберігати ці дані у базі даних MariaDB, а також відстежувати статус роботи датчиків (рис. 3.6). Крім того, реалізована функція очищення таблиці «sensor_data» після накопичення 1000 записів, що допомагає уникнути переповнення бази даних.

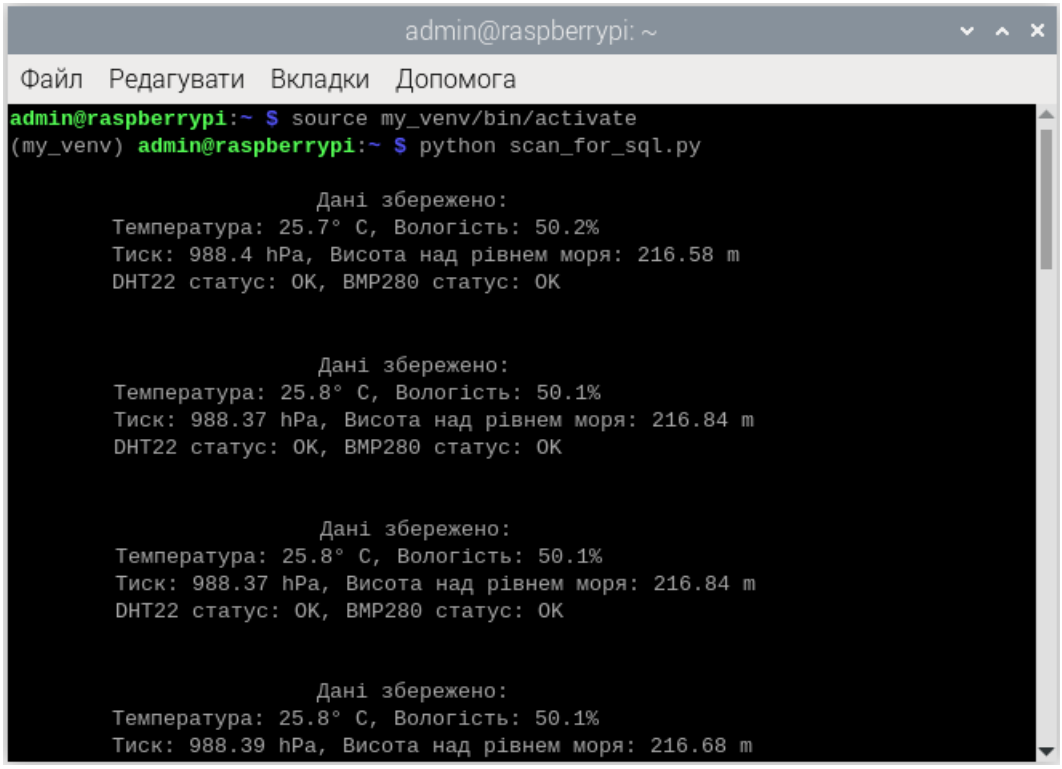

Рисунок 3.6 — Скрипт для зчитування даних з датчиків і збереження у базі даних

Результат роботи Python-скрипта для зчитування даних з датчиків та збереження їх у базі даних MariaDB відображає ефективну інтеграцію сучасних IoT технологій для моніторингу погодних умов. Скрипт успішно з'єднується з датчиками DHT22 та GY-BMP280-3.3, зчитує температуру, вологість, атмосферний тиск та висоту, а потім записує ці дані у таблицю «sensor_data» бази даних «sensors». Це забезпечує надійне зберігання даних та їх доступність для подальшого аналізу.

Для забезпечення стабільної роботи системи зчитування даних з датчиків та їхнього запису до бази даних, використажмо сервісний файл для системи ініціалізації systemd. Цей файл налаштовує службу, яка автоматично запускає скрипт Python під час завантаження системи Raspberry Pi OS. Сервісний файл systemd містить кілька секцій, кожна з яких виконує певну роль:

```
[Unit]
Description=Sensor Read Service
After=network.target
[Service]

Type=simple
User=admin
WorkingDirectory=/home/admin
ExecStart=/bin/bash -c 'source /home/admin/my_venv/bin/activate && exec python3 /home/admin/scan_for_sql.py'
[Install]
WantedBy=multi-user.target
```

Таким чином, наш сервісний файл налаштовує службу, яка запускає скрипт Python scan_for_sql.py всередині віртуального середовища Python після запуску мережевих сервісів.

Таким чином, налаштування служби за допомогою systemd дозволяє ефективно організувати автоматичний запуск та стабільну роботу скрипту.

3.4 Розробка серверу та мобільного програмного забезпечення

У сучасному світі інтеграція серверних і мобільних технологій стала невід'ємною частиною розробки програмного забезпечення. Серверна частина забезпечує централізоване управління даними та логікою застосунка, дозволяючи зберігати, обробляти та передавати дані між клієнтськими застосунками. Це особливо важливо для мобільних застосунків, які взаємодіють з віддаленими серверами для отримання та оновлення інформації в режимі реального часу.

Віддалений сервер виступає центральним вузлом, який обслуговує запити від мобільних застосунків, забезпечуючи доступ до бази даних та інших сервісів. Використання серверів дозволяє підтримувати постійну синхронізацію даних між користувачами, що забезпечує актуальність та консистентність інформації. Сервер також відповідає за автентифікацію користувачів, зберігання великих обсягів даних, обробку складних запитів та інші критично важливі функції, які неможливо реалізувати безпосередньо на мобільному пристрої через обмежені ресурси.

Мобільний застосунок, у свою чергу, забезпечує зручний інтерфейс для користувачів, дозволяючи їм взаємодіяти з сервером і отримувати необхідні дані у будь-який час та будь-якому місці. Використання мобільного ПЗ значно підвищує доступність сервісів і розширює можливості користувачів. Наприклад, мобільний застосунок для моніторингу погодних умов може отримувати дані з віддаленого сервера, який зчитує інформацію з датчиків, і відображати ці дані у реальному часі. Це забезпечує швидкий доступ до важливої інформації та полегшує прийняття рішень на основі актуальних даних.

3.3.1 Розробка сервера віддаленого доступу на базі Flask

Flask є популярним мікрофреймворком для веб-розробки на мові програмування Python. Він забезпечує легкість і гнучкість в розробці, що робить його ідеальним для створення невеликих та середніх веб-застосунків. Деякі з основних переваг Flask:

— простота та легкість, Flask має мінімалістичний дизайн, що дозволяє розробникам швидко розпочати роботу без зайвої конфігурації;

— гнучкість, Flask не нав'язує структуру проєкту, що дозволяє розробникам організовувати код на свій розсуд;

— розширюваність, завдяки модульній архітектурі, Flask легко інтегрується з різними бібліотеками та розширеннями;

— активна спільнота, Flask має велику спільноту користувачів, що забезпечує широкий доступ до ресурсів, документації та підтримки.

Вибір Flask для створення серверу був обґрунтованим рішенням завдяки його простоті, гнучкості та можливості швидкої розробки. Це дозволяє зосередитись на реалізації функціоналу, не витрачаючи багато часу на налаштування [24, 25].

Першим нашим кроком буде – встановлення Flask у віртуальне середовище. Це забезпечує ізольоване середовище для проєкту, що дозволяє уникнути конфліктів між бібліотеками.

```
source my_venv/bin/activate
pip install Flask mysql-connector-python
```

Створимо файл app.py та реалізуємо в ньому простий сервер на Flask, який з'єднується з базою даних MariaDB і повертає останні дані з таблиці sensor_data.

```
from flask import Flask, jsonify
import mysql.connector

import logging
app = Flask(__name__)
# Конфігурація підключення до бази даних
db_config = {
    'user': 'root',
    'password': 'rtpass',
    'host': 'localhost',
    'database': 'sensors',
}
@app.route('/sensors', methods=['GET'])
def get_sensor_data():
    try:
        db_connection = mysql.connector.connect(**db_config)
        cursor = db_connection.cursor(dictionary=True)
        cursor.execute("SELECT * FROM sensor_data ORDER BY timestamp
DESC LIMIT 1")
        sensor_data = cursor.fetchone()
        cursor.close()
        db_connection.close()
        if sensor_data:
```

```
return jsonify(sensor_data), 200
        else:
            return jsonify({"error": "No data found"}), 404
    except Exception as e:
        logging.error(f"Error accessing database: {e}")
        return jsonify({"error": "Internal server error"}), 500
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Наступний крок — налаштування mDNS (Multicast DNS), що дозволяє автоматично знаходити сервіси у локальній мережі без необхідності вказувати IP-адреси. Для налаштування mDNS на Raspberry Pi використовуємо Avahi-daemon.

Встановлюємо через термінал Raspberry Pi.

```
sudo apt-get update
sudo apt-get install avahi-daemon
```

Створюємо файл для конфігурації Avahi.

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
    <name replace-wildcards="yes">%h</name>
    <service>
        <type>_weatherstation._tcp</type>
        <port>5000</port>
    </service>
</service-group>
```

Виконуємо перезапуск Avahi-daemon командою в терміналі.

```
sudo systemctl restart avahi-daemon
```

Ця конфігурація дозволяє Raspberry Pi рекламувати сервер у локальній мережі Wi-Fi, що постійно оновлює дані, зчитувані з датчиків. Завдяки mDNS, інші пристрої в мережі зможуть знайти цей сервер за допомогою імені, а не IP-адреси.

Для забезпечення безперервної роботи веб-сервера Flask, використаємо сервісний файл systemd, який автоматично запускає сервер після завантаження системи. Це дозволяє гарантувати доступність сервера та забезпечити його стабільну роботу без необхідності ручного втручання. Нижче наведено код сервісного файлу та детальний опис його функціональності.

```
[Unit]
Description=Start app.py after boot
After=multi-user.target
[Service]
Type=simple
User=root
WorkingDirectory=/home/admin
ExecStartPre=/bin/sleep 50
ExecStart=/home/admin/my_venv/bin/python /home/admin/app.py
[Install]
WantedBy=multi-user.target
```

Автоматичний запуск сервера Flask за допомогою systemd має кілька важливих переваг. По-перше, він забезпечує безперервну доступність веб-застосунку, гарантуючи його автоматичний запуск після кожного завантаження системи, що зменшує час простою і підвищує надійність сервісу. По-друге, автоматизація запуску значно спрощує адміністрування, зменшуючи необхідність ручного втручання після перезавантаження системи або у разі збоїв, що знижує ризик людських помилок. Крім того, використання systemd надає зручні інструменти для контролю та моніторингу роботи сервісу, такі як перегляд журналів і автоматичний перезапуск у разі проблем. Ізоляція скрипту у віртуальному середовищі Python забезпечує стабільність та відтворюваність роботи веб-сервера, ізолюючи його залежності від решти системи

3.3.2 Розробка мобільного застосунку на базі Xamarin

Мова програмування C# (C-Sharp) – це мова програмування, розроблена компанією Microsoft як частина їхньої платформи .NET. Вона є сучасною, об'єктно-

орієнтованою та типобезпечною мовою, яка поєднує в собі простоту та ефективність. Однією з головних переваг C# є його потужний синтаксис, який підтримує розробку складних застосунків з високою продуктивністю. Крім того, C# має розвинуту стандартну бібліотеку, що значно полегшує розробку та інтеграцію різноманітних функцій, таких як робота з базами даних, мережею, графікою тощо. Також, завдяки активній підтримці Microsoft, мова постійно оновлюється, додаючи нові можливості та покращення. Для розробки мобільного програмного забезпечення було обрано середовище Visual Studio, яке забезпечує потужний інструментарій та зручний інтерфейс для ефективної роботи з кодом [26, 27].

Xamarin — це фреймворк для крос-платформної розробки мобільних застосунків, що дозволяє створювати нативні застосунки для Android, iOS та Windows за допомогою єдиного коду на мові C#. Однією з головних переваг Xamarin є можливість повторного використання до 90% загального коду, що значно зменшує час та витрати на розробку. Xamarin забезпечує повний доступ до нативних API та можливостей платформ, що дозволяє створювати ПЗ з високою продуктивністю та нативним інтерфейсом [28, 29].

Фреймворк також включає в себе Xamarin.Forms, бібліотеку, яка дозволяє розробляти користувацький інтерфейс, що може бути спільним для всіх платформ. Це особливо корисно для застосунів, які мають однаковий або подібний дизайн на різних платформах. Xamarin інтегрується з Visual Studio, що надає розробникам доступ до потужних інструментів для налагодження, тестування та розгортання застосунків.

Мобільне ПЗ, розроблене на Xamarin з використанням мови C# в середовищі Visual Studio 2022, складається з кількох сторінок, кожна з яких виконує певну функцію. ПЗ починається зі стартової сторінки, де відбувається ініціалізація системи та перевірка збережених даних про з'єднання. Якщо дані про з'єднання збережено, користувач переходить до головного меню, де може обрати між інструкцією або переглядом даних метеостанції [30].

Файл InstructionForUser.xaml визначає інтерфейс для сторінки з інструкціями, а файл InstructionForUser.xaml.cs реалізує її логіку. Сторінка містить простий контент, який надає користувачеві необхідну інформацію про використання застосунку (рис. 3.7).

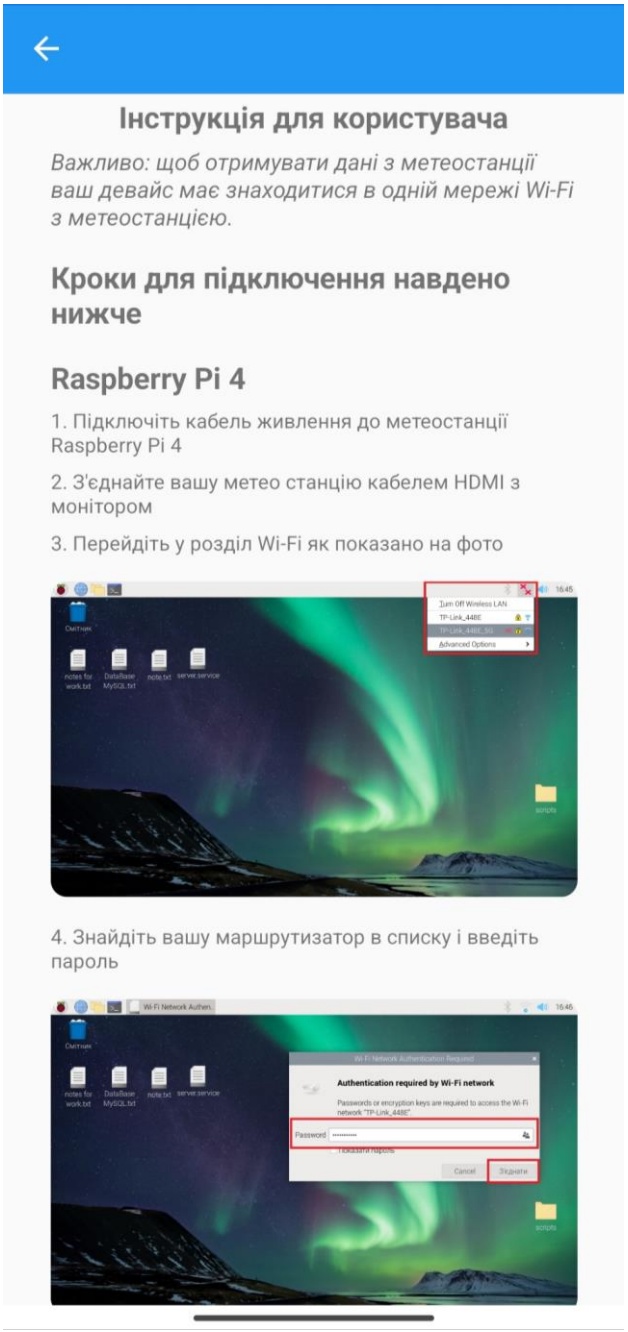

Рисунок 3.7 — Вигляд реалізованої сторінки InstructionForUser.xaml

Файл головної сторінки MainPage.xaml визначає інтерфейс головної сторінки застосунку, а файл MainPage.xaml.cs містить логіку цієї сторінки. При ініціалізації

головної сторінки перевіряється попереднє з'єднання.

```
    private async Task CheckPreviousConnection()
    {
        bool wasConnected =
Application.Current.Properties.ContainsKey(IsConnectedKey) &&

(bool)Application.Current.Properties[IsConnectedKey];
        if (wasConnected &&
Application.Current.Properties.TryGetValue("SensorDataUrl", out var
sensorDataUrl))
        {
            bool answer = await DisplayAlert("Підключення", "Ви вже
підключені до метеостанції. Продовжити сеанс?", "Так", "Ні");
            if (answer)
            {
                await Navigation.PushAsync(new
StationDataPage(sensorDataUrl as string));
            }
            else
            {
                Application.Current.Properties[IsConnectedKey] = false;
                Application.Current.Properties["SensorDataUrl"] = null;
                await Application.Current.SavePropertiesAsync();
            }
        }
    }
```

На головній сторінці також є кнопка для переходу на сторінку з інструкціями та кнопка для пошуку метеостанції у локальній мережі за допомогою Mdns [31, 32].

Після успішного пошуку метеостанції, застосунок пропонує користувачеві підключитися до неї, користувачу необхідно натиснути на кнопку «Почати пошук» (рисунок 3.8) після чого спрацює логіка пошуку і підключення до віддаленого серверу, як показано на лістингу нижче.

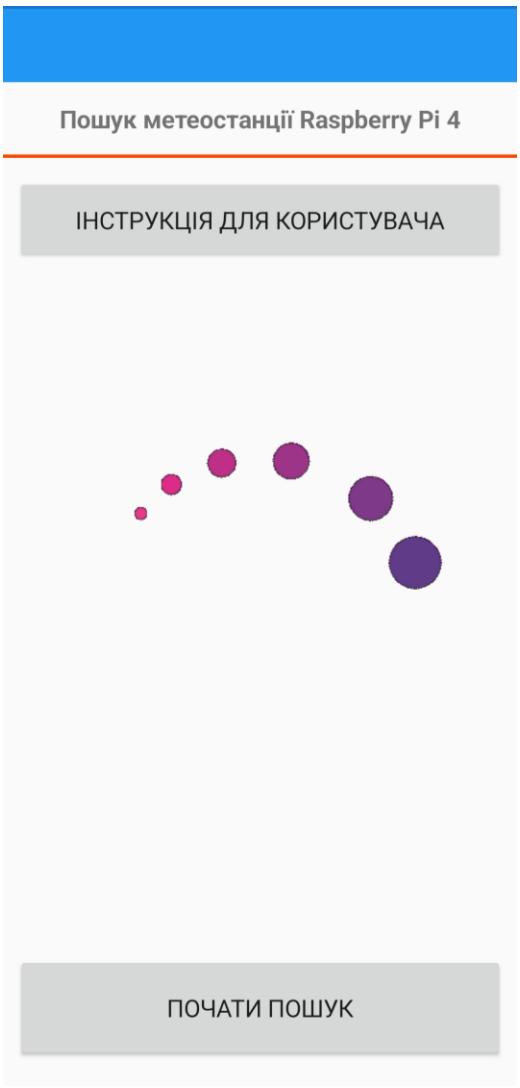

Рисунок 3.8 — Реалізація головної сторінки

```
    private async void OnSearchButtonClicked(object sender, EventArgs e)
    {
        try
        {
            var results = await
ZeroconfResolver.ResolveAsync(ServiceType);
            var service = results.FirstOrDefault();
            if (service != null)
            {
                var address = service.IPAddresses.FirstOrDefault();
                var port = service.Services.FirstOrDefault().Value.Port;
                var sensorDataUrl = $"http://{address}:{port}/sensors";
                Application.Current.Properties[IsConnectedKey] = true;
```

```
                Application.Current.Properties["SensorDataUrl"] =
sensorDataUrl;
                await Application.Current.SavePropertiesAsync();
                bool answer = await DisplayAlert("Успішно знайдено!",
$"Метеостанція знайдена за адресою: {sensorDataUrl}. Бажаєте з'єднатися?",
"З'єднатися", "Відхилити");
                if (answer)
                {
                    await Navigation.PushAsync(new
StationDataPage(sensorDataUrl));
                }
            }
            else
            {
                await DisplayAlert("Помилка", "Не вдалося знайти
метеостанцію.", "OK");
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine($"Помилка: {ex.Message}");
            await DisplayAlert("Помилка", "Виникла проблема з
підключенням до мережі.", "OK");
        }
    }
```

Результат знаходження серверу метеостанції (рис. 3.9).

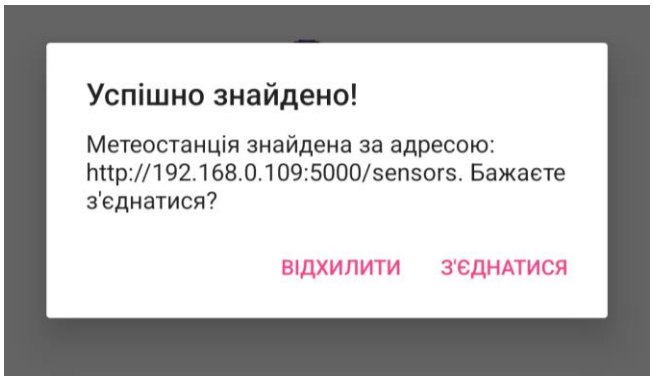

Рисунок 3.9 — Результат знаходження серверу метеостанції

Файл сторінки даних метеостанції StationDataPage.xaml визначає інтерфейс сторінки, яка відображає дані з метеостанції, а файл StationDataPage.xaml.cs містить логіку цієї сторінки. Коли сторінка ініціалізується, вона починає отримувати дані з сервера, реалізованого на Flask, і оновлює інтерфейс (див. рис. 3.10) у реальному часі за допомогою коду:

```
private async Task UpdateSensorDataAsync()
{
    try
    {
        var response = await httpClient.GetAsync(SensorDataUrl);
        if (response.IsSuccessStatusCode)
        {
            var content = await response.Content.ReadAsStringAsync();
            var sensorData =
JsonConvert.DeserializeObject<SensorData>(content);
            Device.BeginInvokeOnMainThread(() =>
UpdateUI(sensorData));
        }
        else
        {
            UpdateFailureCounters();
        }
    }
    catch
    {
        UpdateFailureCounters();
    }
}
```

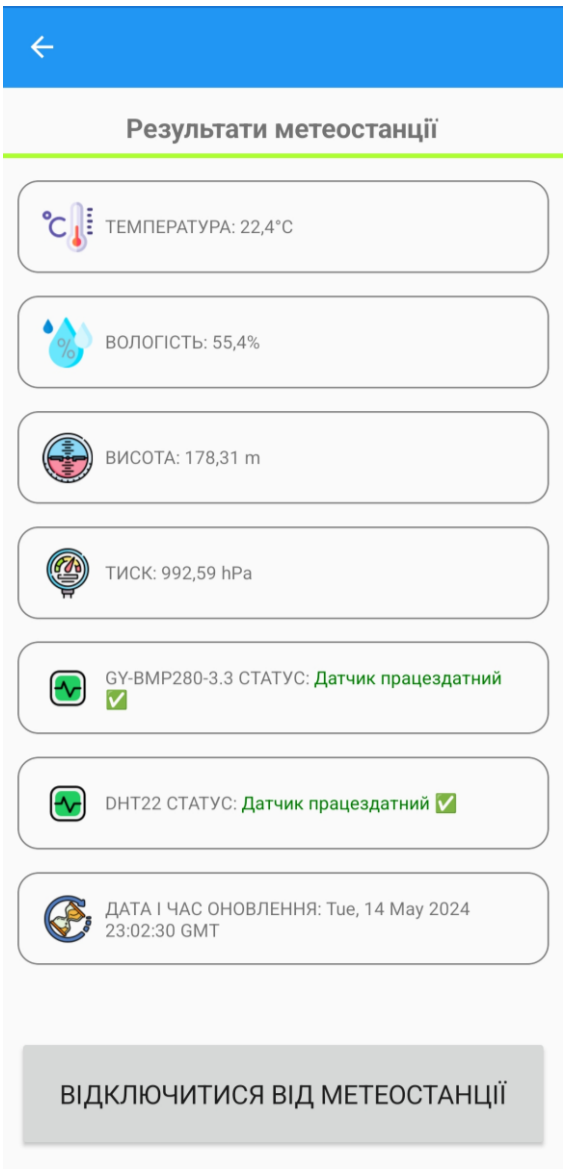

Рисунок 3.10 — Відображення отримання даних з серверу

Дані оновлюються кожну секунду за допомогою таймера, логіка якого реалізована в коді:

```
private void StartRealTimeUpdates()
{
    Device.StartTimer(TimeSpan.FromSeconds(1), () =>
    {
        Task.Run(UpdateSensorDataAsync);
        return true;
    });
}
```

Якщо датчик виходить з ладу, інтерфейс оновлюється відповідно (див. рис 3.11), щоб попередити користувача:

```
private void UpdateDHT22Status(bool hasData)
{
    if (hasData)
    {
        dht22_statusLabel.FormattedText = CreateStatusText("DHT22 СТАТУС:", " Датчик працездатний ☑", Color.Green);

        failedTemperatureUpdatesCounter = 0;
    }
    else
    {
        if (++failedTemperatureUpdatesCounter >= 45)
        {
            dht22_statusLabel.FormattedText = CreateStatusText("DHT22 СТАТУС:", " Увага! Ваш датчик DHT22 вийшов з ладу, необхідно замінити ❌", Color.Red);
        }
    }
}
```

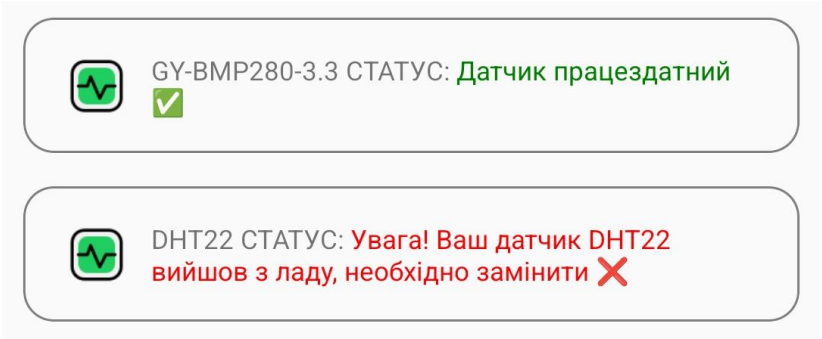

Рисунок 3.11 — Приклад оновлення інтерфейсу з непрацюючим датчиком

Таким чином, мобільний застосунок забезпечує постійний моніторинг стану метеостанції та своєчасне оновлення даних у реальному часі, а також надає користувачеві зручний інтерфейс для взаємодії з системою.

Висновки

У процесі виконання бакалаврського кваліфікаційного проєкту на тему «Мікрокомп’ютерна погодна станція засобами IoT» були досягнуті наступні результати.

Розроблено мікрокомп’ютерну погодну станцію на базі Raspberry Pi 4 Model B. Ця станція дозволяє здійснювати точний і оперативний моніторинг ключових погодних параметрів, таких як температура, вологість і атмосферний тиск. Завдяки використанню сучасних датчиків DHT22 і GY-BMP280-3.3, система забезпечує високу точність вимірювань і надійність даних.

Інтегровано сенсори DHT22 і GY-BMP280-3.3 з мікрокомп’ютером Raspberry Pi 4. Це забезпечило збирання даних про температуру, вологість та атмосферний тиск у реальному часі. Використання бібліотек pigpio, smbus2 та bmp280 дозволило ефективно зчитувати та обробляти дані з сенсорів.

Розроблено сервер на Flask для обробки та зберігання даних. Сервер забезпечує доступ до даних через REST API, що дозволяє легко інтегрувати його з іншими системами та застосунками. Підтримка mDNS дозволяє автоматично знаходити сервер у локальній мережі, що спрощує налаштування та використання системи.

Створено мобільний застосунок на Xamarin для відображення даних з метеостанції. Він дозволяє користувачам отримувати дані про погодні умови у реальному часі, що підвищує зручність використання системи. Застосунок має інтуїтивно зрозумілий інтерфейс і надає можливість віддаленого доступу до даних метеостанції.

Проведено тестування та валідацію системи в реальних умовах. Результати тестувань показали високу точність та надійність системи, що підтверджує її готовність до практичного використання. Система виявилася стійкою до змін погодних умов і забезпечила стабільну роботу протягом усього періоду тестування;

Забезпечено зберігання даних у реляційній БД MariaDB. Це дозволяє зберігати великі обсяги даних та легко їх аналізувати. Використання бази даних

забезпечує можливість довготривалого зберігання історичних даних та їх подальший аналіз.

Цей проєкт має значний потенціал для подальшого вдосконалення та розширення функціоналу. Систему можна доповнити додатковими датчиками для моніторингу інших параметрів навколишнього середовища, наприклад, для детектування шкідливих газів або відстеження руху в приміщенні.

Результати даної роботи підтверджують можливість ефективного використання мікрокомп’ютерів та IoT-технологій для моніторингу погодних умов. Система є надійною, точною та доступною, що робить її привабливим інструментом для широкого кола користувачів.

Список використаних джерел

1. Метрологія: предмет, структура та функції, міжнародні організації: вебсайт. URL: https://osvita.ua/vnz/reports/econom_pidpr/19085/ (дата звернення 12.03.2024).

2. Технології індустрії 4.0: вебсайт. URL: http://edu.asu.in.ua/mod/book/view.php?id=112 (дата звернення 14.03.2024).

3. Технології інтернету речей: вебсайт. URL: https://ela.kpi.ua/server/api/core/bitstreams/dcd9e1aa-8bcc-4e76-b1e0-ed133bf616b2/content (дата звернення 14.03.2024).

4. Які функції виконують метеостанції: основні завдання та обладнання: вебсайт. URL: https://ideas.woodstar.com.ua/yaki-funkcii-vikonuyut-meteostancii-osnovni-zavdannya-ta-obladnannya/ (дата звернення 18.03.2024).

5. Метеорологічна станція: вебсайт. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B5%D0%BE%D1%80%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B0_%D1%81%D1%82%D0%B0%D0%BD%D1%86%D1%96%D1%8F (дата звернення 20.03.2024).

6. Метеостанції для дому: вебсайт. URL: https://comfortshop.com.ua/blog/meteostantsii-dlya-doma/ (дата звернення 22.03.2024).

7. Road Weather Station: вебсайт. URL: https://www.darrera.com/en/product/3r-aws300-road-weather-station/ (дата звернення 23.03.2024).

8. Діаграми UML для моделювання процесів і архітектури: вебсайт. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення 25.03.2024).

9. Online UML Diagram Tool: вебсайт. URL: https://www.smartdraw.com/uml-diagram/uml-diagram-tool.htm (дата звернення 25.03.2024).

10. Fritzing Electronics Made Easy: вебсайт. URL: https://fritzing.org/learning/ (дата звернення 27.03.2024).

11. Бібліотека елементів для програми Fritzing IDE: вебсайт. URL: https://github.com/adafruit/Fritzing-Library/blob/master/parts/Adafruit%20BMP180.fzpz (дата звернення 27.03.2024).

12. Xamarin – Android Activities: вебсайт. URL: https://www.codejourney.net/xamarin-android-activities/ (дата звернення 29.03.2024).

13. Arduino проти Raspberry Pi: яка плата найкраща?: вебсайт. URL: https://uk.fmuser.net/content/?10733.html (дата звернення 31.03.2024).

14. DHT11 vs DHT22 Sensors: вебсайт. URL: https://learn.adafruit.com/dht (дата звернення 31.03.2024).

15. Датчик вологості та температури DHT21/AM2301A: вебсайт. URL: https://arduino.ua/ru/prod1683-datchik-vlajnosti-i-temperatyri-dht21am2301 (дата звернення 01.04.2024).

16. Модуль GY-BMP280-3.3 високоточний датчик атмосферного тиску – висотомір: вебсайт. URL: https://electronica.in.ua/ua/p1530391848-modul-bmp280-vysokotochnyj.html (дата звернення 02.04.2024).

17. BMP085 Digital Pressure Sensor: вебсайт. URL: https://learn.adafruit.com/bmp085 (дата звернення 02.04.2024).

18. Модуль GY-68 BMP180 датчик температури і тиску BOSCH: вебсайт. URL: https://electronica.in.ua/ua/p1530391847-modul-bmp180-datchik.html (дата звернення 02.04.2024).

19. Raspberry Pi 4 Faster, more powerful: вебсайт. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (дата звернення 03.04.2024).

20. Tutorial 26: DHT22 Digital Temp / Humidity Sensor: вебсайт. URL: https://thezanshow.com/electronics-tutorials/raspberry-pi/tutorial-26#:~:text=import%20pigpio%20import%20DHT22%20from%20time%20import%20sleep,27%29%20%23%20use%20the%20actual%20GPIO%20pin%20name (дата звернення 04.04.2024).

21. Configuring BMP280 Sensor with Raspberry Pi: вебсайт. URL: https://iotstarters.com/configuring-bmp280-sensor-with-raspberry-pi/ (дата звернення 06.04.2024).

22. SQL Syntax: вебсайт. URL: https://www.w3schools.com/sql/sql_syntax.asp (дата звернення 10.04.2024).

23. MariaDB Tutorial: вебсайт. URL: https://www.mariadbtutorial.com/ (дата звернення 10.04.2024).

24. Tutorial – Flask Documentation: вебсайт. URL: https://flask.palletsprojects.com/en/3.0.x/tutorial/ (дата звернення 00.00.2024).

25. Мега-Підручник Flask: вебсайт. URL: https://habr.com/en/articles/804245/ (дата звернення 11.04.2024).

26. Visual Studio Community: вебсайт. URL: https://visualstudio.microsoft.com/vs/community/ (дата звернення 16.04.2024).

27. C# Tutorial: вебсайт. URL: https://www.w3schools.com/cs/index.php (дата звернення 16.04.2024).

28. Що таке Xamarin?: вебсайт. URL: https://learn.microsoft.com/uk-ua/previous-versions/xamarin/get-started/what-is-xamarin (дата звернення 17.04.2024).

29. Платформа Xamarin.Android. Розробка мобільних програм за допомогою мови C#: вебсайт. URL: https://itvdn.com/ua/channel/video/webinar-platform-xamarin-android (дата звернення 20.04.2024).

30. Xamarin.Android Application Fundamentals: вебсайт. URL: https://learn.microsoft.com/uk-ua/previous-versions/xamarin/android/app-fundamentals/ (дата звернення 22.04.2024).

31. Multicast DNS (MDNS) on Home Networks: вебсайт. URL: https://stevessmarthomeguide.com/multicast-dns/ (02.05.2024).

32. Find the MQTT broker without an IP address: вебсайт URL: https://dagrende.blogspot.com/2017/02/find-mqtt-broker-without-hard-coded-ip.html (дата звернення 03.05.2024).

Додаток А
Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д.

« 06 » травня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврського дипломного проєкту

«Мікрокомп’ютерна погодна станція засобами IoT»

08-54.БДП.003.00.000 ПЗ

Науковий керівник: доцент к.т.н.

_____ Богомолов С.В.

Студент групи КІ-22мсз

_____ Морозов П.В.

ВНТУ 2024

1. Підставою для виконання бакалаврського дипломного проєкту (БДП)

1.1 Актуальність дослідження обумовлена необхідністю створення мікрокомп’ютерної погодної станції засобами IoT, яка здатна забезпечити точний та оперативний моніторинг погодних умов у реальному часі. Застосування Raspberry Pi 4 з сенсорами DHT22 та GY-BMP280-3.3 дозволяє отримувати дані про температуру, вологість, висоту та атмосферний тиск, що є важливим для багатьох сфер діяльності, включаючи агрономію, будівництво, екологію та інші;

1.2 Наказ про затвердження тема БДП.

2. Мета БДП і призначення розробки

2.1 Мета роботи – розробка мікрокомп’ютерної погодної станції на основі Raspberry Pi 4, яка забезпечить точне та оперативне вимірювання погодних умов;

2.2 Призначення розробки – система моніторингу погодних умов, призначена для отримання, обробки та аналізу даних про температуру, вологість та атмосферний тиск у реальному часі. Система включає інтеграцію сенсорів DHT22 і GY-BMP280-3.3, серверну частину на Flask та мобільний застосунок на Xamarin для віддаленого доступу до даних

3. Вихідні дані для виконання БДП

3.1 Проведення аналізу існуючих рішень у сфері метеорології та технологій Інтернету речей (IoT);

3.2 Розробка структури та функціональної схеми мікрокомп’ютерної погодної станції;

3.3 Інтеграція сенсорів DHT22 і GY-BMP280-3.3 з мінікомп'ютером Raspberry Pi 4;

3.4 Розробка серверної частини на Flask для обробки та зберігання даних;

3.5 Створення мобільного застосунку на Xamarin для моніторингу даних у реальному часі;

3.6 Проведення тестування та валідації системи в реальних умовах.

4. Вимоги до виконання БДП

4.1 Використання сенсорів DHT22 і GY-BMP280-3.3 для збору даних про

погодні умови;

4.2 Розробка програмного забезпечення на мові Python для зчитування та обробки даних з сенсорів;

4.3 Розробка веб-сервера на Flask для зберігання даних БД MariaDB;

4.4 Створення мобільного застосунку на Xamarin для віддаленого моніторингу даних;

4.5 Забезпечення автоматичного запуску скриптів та сервісів за допомогою systemd;

4.6 Проведення тестування системи для забезпечення її надійності та стабільності в умовах експлуатації.

Розробка мікрокомп'ютерної погодної станції з використанням IoT-технологій, яка включає в себе інтеграцію датчиків DHT22 і GY-BMP280-3.3 з мінікомп'ютером Raspberry Pi 4 Model B, сервер на Flask та мобільний застосунок на Xamarin для моніторингу даних у реальному часі

5. Етапи БДП та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1

Таблиця А.1 – Етапи БКП

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз літературних джерел. Попередня розробка основних розділів	12.03.2024	20.03.2024	Вступ
2	Огляд та аналіз існуючих рішень для проєктування метеорологічних станцій	26.03.2024	01.04.2024	Розділ 1
3	Розробка мікрокомп'ютерної погодної станції: апаратна частина та мобільний інтерфейс	02.04.2024	07.04.2024	Розділ 2

Продовження таблиці А.1 – Етапи БКП

4	Програмна реалізація метеостанції та розробка Android-застосунку	08.04.2024	10.05.2024	Розділ 3
5	Пояснювальна записка	11.05.2024	19.05.2024	Презентація

6. Матеріали, що подаються до захисту БДП

До захисту подаються: пояснювальна записка БДП, графічні матеріали, протокол попереднього захисту БКП на кафедрі, відгук наукового керівника, рецензію, анотації до БДП українською та іноземною мовами, довідка про відповідність оформлення БДП діючим вимогам.

7. Порядок контролю виконання та захисту БДП:

Виконання етапів графічної документації БДП контролюється науковим керівником згідно зі встановленими термінами. Захист БДП відбувається на засіданні Екзаменаційної комісії, затвердженою наказом ректора.

8. Вимоги до оформлювання та порядок виконання БДП

8.1 При оформлюванні БДП використовуються:

– ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

– ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

– міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

– методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2024;

– документами на які посилаються у вище вказаних.

8.2 Порядок виконання БДП викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіт СУЯ ВНТУ-03.02.02-П.001.01:21».

Додаток Б

Лістинг коду датчиків DHT22 і DY-BMP280-3.3

Лістинг для датчика DHT22:

```
# Імпортуємо бібліотеку pigpio для роботи з GPIO
import pigpio
# Імпортуємо бібліотеку pigpio_dht для роботи з датчиками DHT
import pigpio_dht
# Імпортуємо стандартний модуль time для затримки
import time
# Задаємо номер GPIO-піна, до якого підключений датчик
GPIO_PIN = 4
# Створюємо екземпляр датчика DHT22, вказуючи номер GPIO-піна
dht_sensor = pigpio_dht.DHT22(GPIO_PIN)
# Створюємо екземпляр pigpio для роботи з GPIO
pi = pigpio.pi()
# Перевіряємо, чи встановлено з'єднання з pigpio
if not pi.connected:
    # Якщо не встановлено, виходимо з програми
    exit()
try:
    # Нескінченний цикл для постійного зчитування даних
    while True:
        # Зчитуємо дані з датчика
        result = dht_sensor.read()
        # Виводимо значення температури та вологості у форматі:
        print(f"Temperature: {result['temp_c']} Deg C, Humidity:
{result['humidity']}%")
        # Затримка на 2 секунди перед наступним циклом
        time.sleep(2)
finally:
    # Зупиняємо роботу pigpio перед завершенням програми
    pi.stop()
```

Лістинг для датчика GY-BMP280-3.3:

```
# Імпортуємо бібліотеку smbus2 для взаємодії з інтерфейсом I2C
import smbus2
# Імпортуємо бібліотеку bmp280 для роботи з датчиком BMP280
import bmp280
# Імпортуємо стандартний модуль time для затримки
import time
# Задаємо номер I2C порту, до якого підключений датчик
I2C_PORT = 1
# Задаємо адресу датчика BMP280 на шині I2C
BMP280_ADDRESS = 0x76
# Створюємо екземпляр SMBus для роботи з I2C
bus = smbus2.SMBus(I2C_PORT)
# Створюємо екземпляр датчика BMP280
sensor = bmp280.BMP280(i2c_dev=bus, i2c_addr=BMP280_ADDRESS)
# Нескінченний цикл для постійного зчитування даних
while True:
    # Зчитуємо значення атмосферного тиску
    pressure = sensor.get_pressure()
    # Зчитуємо значення висоти над рівнем моря
    altitude = sensor.get_altitude()
    # Виводимо значення тиску і висоти
    print(f"Pressure: {pressure:.2f} hPa, Altitude: {altitude:.2f} m")
```

```
    # Затримка на 2 секунди перед наступним циклом
    time.sleep(2)
```

Лістинг для датчиків DHT22 і GY-BMP280-3.3 з можливістю збереження даних до БД:

```
# Імпортування необхідних бібліотек
import smbus2
import bmp280
import pigpio
import pigpio_dht
import time
import mysql.connector
import logging
# Налаштування логування
logging.basicConfig(filename='sensor_data.log', level=logging.INFO)
# Задання констант
I2C_PORT = 1
BMP280_ADDRESS = 0x76
# Ініціалізація датчика BMP280
bus = smbus2.SMBus(I2C_PORT)
sensor = bmp280.BMP280(i2c_dev=bus, i2c_addr=BMP280_ADDRESS)
# Задання номеру GPIO-піна для датчика DHT22
GPIO_PIN = 4
# Ініціалізація датчика DHT22
dht_sensor = pigpio_dht.DHT22(GPIO_PIN)
pi = pigpio.pi()
# Параметри для підключення до бази даних
db_config = {
 'user': 'root',
 'password': 'rtpass',
 'host': 'localhost',
 'database': 'sensors',
 'raise_on_warnings': True
}
try:
   # Підключення до бази даних
   db_connection = mysql.connector.connect(**db_config)
   cursor = db_connection.cursor()
   # Перевірка з'єднання з pigpio
   if not pi.connected:
       exit()
   while True:
       # Перевірка датчика BMP280
       try:
           pressure = round(sensor.get_pressure(), 2)  # Округлення до
двох знаків після коми
           altitude = round(sensor.get_altitude(), 2)  # Округлення до
двох знаків після коми
           bmp280_status = "OK"
           logging.info("Info Save")
       except Exception as e:
           print(f"Помилка зчитування з BMP280: {e}")
           bmp280_status = "FAIL"
           pressure, altitude = None, None  # Встановлюємо значення як
None, якщо зчитування не вдалося
           logging.error("Error")
       # Перевірка датчика DHT22
       result = dht_sensor.read()
       dht22_status = "OK" if result['valid'] else "FAIL"
```

```
        # Запис даних у базу даних
        insert_query = ("INSERT INTO sensor_data (temperature,
humidity, pressure, altitude, dht22_status, bmp280_status) "
                        "VALUES (%s, %s, %s, %s, %s, %s)")
        data_tuple = (result['temp_c'] if result['valid'] else None,
                      result['humidity'] if result['valid'] else None,
                      pressure, altitude, dht22_status, bmp280_status)
        cursor.execute(insert_query, data_tuple)
        db_connection.commit()
        # Перевірка кількості записів і очищення таблиці, якщо потрібно
        cursor.execute("SELECT COUNT(*) FROM sensor_data")
        count = cursor.fetchone()[0]
        if count >= 1000:
            cursor.execute("TRUNCATE TABLE sensor_data")
            db_connection.commit()
            logging.info("Table truncated due to reaching 1000
records.")
        print(f"\n\t\t\tДані збережено: \n\tТемпература:
{result['temp_c'] if result['valid'] else 'N/A'}° C, Вологість:
{result['humidity'] if result['valid'] else 'N/A'}% \n\tТиск:
{pressure if pressure is not None else 'N/A'} hPa, Висота над рівнем
моря: {altitude if altitude is not None else 'N/A'} m \n\tDHT22
статус: {dht22_status}, BMP280 статус: {bmp280_status}\n")
        time.sleep(2.0)
finally:
    # Зупинка роботи pigpio
    pi.stop()
    cursor.close()
    db_connection.close()
```

Лістинг сервісного файлу для автоматичного запуску скрипта:

```
[Unit]
Description=Sensor Read Service
After=network.target
[Service]
Type=simple
User=admin
WorkingDirectory=/home/admin
ExecStart=/bin/bash -c 'source /home/admin/my_venv/bin/activate &&
exec python3 /home/admin/scan_for_sql.py'
[Install]
WantedBy=multi-user.target
```

Додаток В

База даних для зберігання метеорологічних показників

Вихідний код створення бази даних:

```
CREATE DATABASE sensors;
```

Вихідний код таблиці бази даних, для збереження даних показників:

```
CREATE TABLE sensor_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    temperature DECIMAL(5,2),
    humidity DECIMAL(5,2),
    pressure DECIMAL(7,2),
    altitude DECIMAL(7,2),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    dht22_status VARCHAR(50),
    bmp280_status VARCHAR(50)
);
```

Додаток Г

Лістинг коду веб-серверу для моніторингу погодних даних

Лістинг файлу app,py реалізації серверу:

```
from flask import Flask, jsonify
import mysql.connector
import logging

app = Flask(__name__)
# Конфігурація підключення до бази даних
db_config = {
    'user': 'root',
    'password': 'rtpass',
    'host': 'localhost',
    'database': 'sensors',
}
# Визначаємо маршрут для обробки GET-запитів
@app.route('/sensors', methods=['GET'])
def get_sensor_data():
    try:
        # Встановлюємо з'єднання з базою даних
        db_connection = mysql.connector.connect(**db_config)
        # Створюємо курсор для виконання запитів
        cursor = db_connection.cursor(dictionary=True)
        # Виконуємо запит для отримання останнього запису з таблиці
sensor_data
        cursor.execute("SELECT * FROM sensor_data ORDER BY timestamp
DESC LIMIT 1")
        # Отримуємо результат запиту
        sensor_data = cursor.fetchone()
        # Закриваємо курсор
        cursor.close()
        # Закриваємо з'єднання з базою даних
        db_connection.close()
        if sensor_data:
            # Якщо є дані, повертаємо їх у форматі JSON з кодом
статусу 200 (OK)
            return jsonify(sensor_data), 200
        else:
            # Якщо даних немає, повертаємо повідомлення про помилку та
код статусу 404 (Not Found)
            return jsonify({"error": "No data found"}), 404
    except Exception as e:
        # Якщо виникла помилка, записуємо її в лог та повертаємо
повідомлення про помилку сервера
        logging.error(f"Error accessing database: {e}")
        return jsonify({"error": "Internal server error"}), 500
if __name__ == '__main__':
    # Запускаємо сервер Flask на всіх доступних мережевих інтерфейсах
('0.0.0.0') та порту 5000
    # Встановлюємо режим debug=True для полегшення налагодження
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Лістинг файлу server.service для рекламування серверу в мережі Wi-Fi:

```
<?xml version="1.0" standalone='no'?><!--*-nxml-*-->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
```

```
    <name replace-wildcards="yes">%h</name>
    <service>
        <type>_weatherstation._tcp</type>
        <port>5000</port>
    </service>
</service-group>
```

Лістинг коду сервісного файлу app.service для автоматичного запуску:

```
[Unit]
Description=Start app.py after boot
After=multi-user.target
[Service]
Type=simple
User=root
WorkingDirectory=/home/admin
ExecStartPre=/bin/sleep 50
ExecStart=/home/admin/my_venv/bin/python /home/admin/app.py
[Install]
WantedBy=multi-user.target
```

Додаток Д

Блок-схема алгоритму застосунку

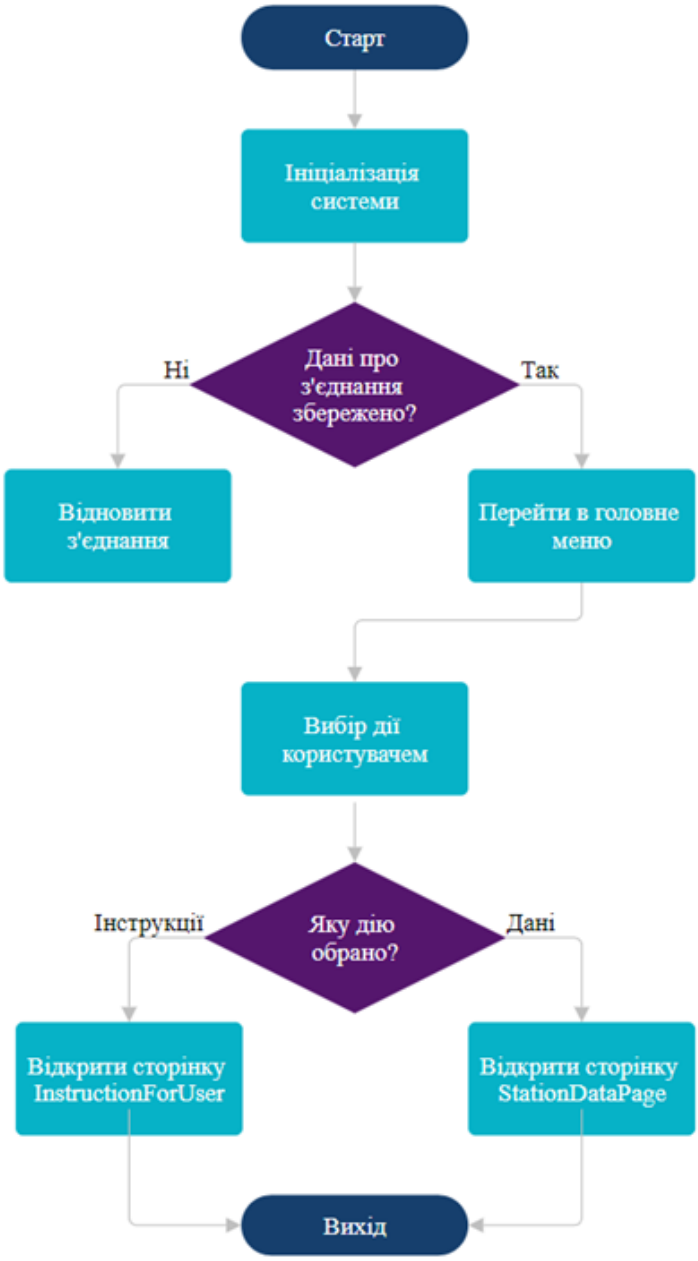

Рисунок Д.1 — Блок-схема алгоритму роботи мобільного застосунку

Додаток Е

Діаграма діяльності розробки метеостанції

Рисунок Е.1 — Діаграма діяльності розробки метеостанції

Додаток Ж

Лістинг коду мобільного застосунку

MainPage.xaml:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:ffimageloading="clr-
namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"
             x:Class="Full_App.MainPage">
    <StackLayout VerticalOptions="FillAndExpand" Spacing="15">
        <!-- Заголовок "Пошук метеостанції Raspberry Pi" -->
        <Label Text="Пошук метеостанції Raspberry Pi 4"
               VerticalOptions="Start"
               HorizontalOptions="CenterAndExpand"
               FontSize="Large"
               FontAttributes="Bold"
               Margin="0,15,0,0"
               Scale="1.0"/>
        <BoxView HeightRequest="2.5"
                 HorizontalOptions="FillAndExpand"
                 Color="OrangeRed"/>
        <!-- Посилання на сторінку з інструкцією -->
        <Button Text="Інструкція для користувача"
                VerticalOptions="End"
                HeightRequest="65"
                FontSize="20"
                Margin="10,0,10,20"
                Clicked="instructionClick"/>
        <!-- Відтворення gif -->
        <ffimageloading:CachedImage Source="loader.gif"
                                    x:Name="animationView"
                                    VerticalOptions="FillAndExpand"
                                    HorizontalOptions="FillAndExpand"
                                    Aspect="AspectFit" />
        <!-- Кнопка "Почати пошук" -->
        <Button Text="Почати пошук"
                VerticalOptions="End"
```

```
                HeightRequest="80"
                FontSize="20"
                Margin="10,0,10,20"
                Clicked="OnSearchButtonClicked"/>
    </StackLayout>
</ContentPage>
```

MainPage.xaml.cs:

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Xamarin.Forms;
using System.Net.Http;
using Zeroconf;
using System.Diagnostics;
namespace Full_App
{
    public partial class MainPage : ContentPage
    {
        private HttpClient httpClient = new HttpClient();
        // Ключ для зберігання статусу підключення в властивостях
застосунку
        public static readonly string IsConnectedKey =
"IsConnectedToStation";
        // Тип сервісу для пошуку метеостанції
        public static readonly string ServiceType =
"_weatherstation._tcp.local.";
        // Конструктор головної сторінки
        public MainPage()
        {
            InitializeComponent();
            CheckPreviousConnection().ConfigureAwait(false);
        }
        // Обробник для кнопки переходу на сторінку інструкції
        private async void instructionClick(object sender, EventArgs
e)
        {
            await Navigation.PushAsync(new InstructionForUser());
        }
        // Обробник для кнопки пошуку метеостанції
```

```
        private async void OnSearchButtonClicked(object sender,

EventArgs e)
        {
            try
            {
                // Виконання пошуку метеостанцій в локальній мережі
                var results = await
ZeroconfResolver.ResolveAsync(ServiceType);
                var service = results.FirstOrDefault();
                if (service != null)
                {
                    // Отримання IP-адреси і порту сервісу
                    var address =
service.IPAddresses.FirstOrDefault();
                    var port =
service.Services.FirstOrDefault().Value.Port;
                    var sensorDataUrl =
$"http://{address}:{port}/sensors";
                    // Збереження даних підключення у властивостях
додатка
                    Application.Current.Properties[IsConnectedKey] =
true;
                    Application.Current.Properties["SensorDataUrl"] =
sensorDataUrl;
                    await Application.Current.SavePropertiesAsync();
                    // Оновлення DisplayAlert і перехід на
StationDataPage з передачею URL
                    bool answer = await DisplayAlert("Успішно
знайдено!", $"Метеостанція знайдена за адресою: {sensorDataUrl}.
Бажаєте з'єднатися?", "З'єднатися", "Відхилити");
                    if (answer)
                    {
                        await Navigation.PushAsync(new
StationDataPage(sensorDataUrl));
                    }
                }
                else
                {
```

```
                    // Відображення повідомлення про помилку, якщо
            сервіс не знайдено
                     await DisplayAlert("Помилка", "Не вдалося знайти
метеостанцію.", "OK");
                }
            }
            catch (Exception ex)
            {
                // Обробка виключень і відображення повідомлення про
помилку
                Debug.WriteLine($"Помилка: {ex.Message}");
                await DisplayAlert("Помилка", "Виникла проблема з
підключенням до мережі.", "OK");
            }
        }
        // Перевірка попереднього підключення до метеостанції
        private async Task CheckPreviousConnection()
        {
             // Перевірка, чи є збережені дані про попереднє
підключення
            bool wasConnected =
Application.Current.Properties.ContainsKey(IsConnectedKey) &&

(bool)Application.Current.Properties[IsConnectedKey];
            if (wasConnected &&
Application.Current.Properties.TryGetValue("SensorDataUrl", out var
sensorDataUrl))
            {
                bool answer = await DisplayAlert("Підключення", "Ви
вже підключені до метеостанції. Продовжити сеанс?", "Так", "Ні");
                if (answer)
                {
                      // Відображення запиту на підтвердження
відновлення підключення
                    await Navigation.PushAsync(new
StationDataPage(sensorDataUrl as string));
                }
                else
                {
```

```
                    // Очищення збережених даних про підключення, якщо
            користувач відмовляється
                    Application.Current.Properties[IsConnectedKey] =
false;
                    Application.Current.Properties["SensorDataUrl"] =
null;
                    await Application.Current.SavePropertiesAsync();
                }
            }
        }
    }
}
```

StationDataPage.xaml:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Full_App.StationDataPage">
    <StackLayout VerticalOptions="FillAndExpand">
        <StackLayout VerticalOptions="StartAndExpand">
            <Label Text="Результати метеостанції"
                   VerticalOptions="Start"
                   HorizontalOptions="CenterAndExpand"
                   FontSize="Large"
                   FontAttributes="Bold"
                   Margin="0,15,0,0"
                   Scale="1.0"/>
            <BoxView HeightRequest="3"
                     HorizontalOptions="FillAndExpand"
                     Color="GreenYellow"/>
            <!-- Блок для температури з зображенням та текстом -->
            <Frame CornerRadius="14"
                   BorderColor="Gray"
                   BackgroundColor="Transparent"
                   Padding="15"
                   Margin="10, 10, 10, 10">
                <StackLayout Orientation="Horizontal"
                             VerticalOptions="Center">
                    <Image Source="centigrade.png"
                           WidthRequest="40"
```

```
                        HeightRequest="35"/>
                    <Label x:Name="temperatureLabel"
Text="TEMPERATURE" VerticalOptions="Center" />
                </StackLayout>
            </Frame>
            <!-- Блок для вологості з зображенням та текстом -->
            <Frame CornerRadius="14"
                   BorderColor="Gray"
                   BackgroundColor="Transparent"
                   Padding="15"
                   Margin="10, 0, 10, 10">
                <StackLayout Orientation="Horizontal"
VerticalOptions="Center">
                    <Image Source="humidity.png" WidthRequest="40"
HeightRequest="35"/>
                    <Label x:Name="humidityLabel" Text="HUMIDITY"
VerticalOptions="Center" />
                </StackLayout>
            </Frame>
            <!-- Блок для висоти над рівнем моря з зображенням та
текстом -->
            <Frame CornerRadius="14"
                   BorderColor="Gray"
                   BackgroundColor="Transparent"
                   Padding="15"
                   Margin="10, 0, 10, 10">
                <StackLayout Orientation="Horizontal"
VerticalOptions="Center">
                    <Image Source="altitude.png" WidthRequest="40"
HeightRequest="35"/>
                    <Label x:Name="altitudeLabel" Text="ALTITUDE"
VerticalOptions="Center" />
                </StackLayout>
            </Frame>
            <!-- Блок для тиску з зображенням та текстом -->
            <Frame CornerRadius="14"
                   BorderColor="Gray"
                   BackgroundColor="Transparent"
                   Padding="15"
```

```
                Margin="10, 0, 10, 10">
                <StackLayout Orientation="Horizontal"
VerticalOptions="Center">
                    <Image Source="gauge.png" WidthRequest="40"
HeightRequest="35"/>
                    <Label x:Name="pressureLabel" Text="PRESSURE"
VerticalOptions="Center" />
                </StackLayout>
            </Frame>
            <!-- Блок для статусу датчика GY-BMP280-3.3 з зображенням
та текстом -->
            <Frame CornerRadius="14"
                   BorderColor="Gray"
                   BackgroundColor="Transparent"
                   Padding="15"
                   Margin="10, 0, 10, 10">
                <StackLayout Orientation="Horizontal"
VerticalOptions="Center">
                    <Image Source="status.png" WidthRequest="40"
HeightRequest="35"/>
                    <Label x:Name="bmp280_statusLabel" Text="GY-
BMP280-3.3 STATUS" VerticalOptions="Center" />
                </StackLayout>
            </Frame>
            <!-- Блок для статусу датчика DHT22 з зображенням та
текстом -->
            <Frame CornerRadius="14"
                   BorderColor="Gray"
                   BackgroundColor="Transparent"
                   Padding="15"
                   Margin="10, 0, 10, 10">
                <StackLayout Orientation="Horizontal"
VerticalOptions="Center">
                    <Image Source="status.png" WidthRequest="40"
HeightRequest="35"/>
                    <Label x:Name="dht22_statusLabel" Text="DHT22
STATUS" VerticalOptions="Center" />
                </StackLayout>
            </Frame>
```

```
            <!-- Блок для отримання дати та часу з зображенням та
текстом -->
            <Frame CornerRadius="14"
                   BorderColor="Gray"
                   BackgroundColor="Transparent"
                   Padding="15"
                   Margin="10, 0, 10, 10">
                <StackLayout Orientation="Horizontal"
VerticalOptions="Center">
                    <Image Source="time_passing.png" WidthRequest="40"
HeightRequest="35"/>
                    <Label x:Name="timestampLabel" Text="TIMESTAMP"
VerticalOptions="Center" />
                </StackLayout>
            </Frame>
        </StackLayout>
        <Button x:Name="disconnectButton"
                Text="Відключитися від метеостанції"
                VerticalOptions="End"
                HeightRequest="80"
                FontSize="20"
                Margin="10,0,10,20"
                Clicked="OnDisconnectButtonClicked"/>
    </StackLayout>
</ContentPage>
```

StationDataPage.xaml.cs:

```
using System;
using System.Threading.Tasks;
using System.Net.Http;
using Newtonsoft.Json;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Full_App
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class StationDataPage : ContentPage
    {
        // створення екземпляру класу HttpClient для надсилання HTTP-
запитів і отримання HTTP-відповідей від веб-сервісу
```

```
        private HttpClient httpClient = new HttpClient();
        // URL для отримання даних
        private string SensorDataUrl;
        // Лічильник невдалих оновлень вологості
        private int failedHumidityUpdatesCounter = 0;
        // Конструктор сторінки з прийманням URL для отримання даних
        public StationDataPage(string sensorDataUrl)
        {
            InitializeComponent();
            SensorDataUrl = sensorDataUrl;
            SetInitialValues();
            StartRealTimeUpdates();
        }
        // Встановлення початкових значень для елементів UI
        private void SetInitialValues()
        {
            temperatureLabel.Text = "ТЕМПЕРАТУРА: Analysis";
            humidityLabel.Text = "ВОЛОГІСТЬ: Analysis";
            pressureLabel.Text = "ТИСК: Analysis";
            altitudeLabel.Text = "ВИСОТА: Analysis";
            bmp280_statusLabel.Text = "GY-BMP280-3.3 СТАТУС:
Analysis";
            dht22_statusLabel.Text = "DHT22 СТАТУС: Analysis";
            timestampLabel.Text = "ДАТА І ЧАС ОНОВЛЕННЯ: Analysis";
        }
        // Початок реальних оновлень даних з датчиків з інтервалом в 1
секунду
        private void StartRealTimeUpdates()
        {
            Device.StartTimer(TimeSpan.FromSeconds(1), () =>
            {
                Task.Run(UpdateSensorDataAsync);
                return true;
            });
        }
        // Асинхронне оновлення даних з датчиків

        private async Task UpdateSensorDataAsync()
        {
```

```
            try
            {
                var response = await
httpClient.GetAsync(SensorDataUrl);
                if (response.IsSuccessStatusCode)
                {
                    var content = await
response.Content.ReadAsStringAsync();
                    var sensorData =
JsonConvert.DeserializeObject<SensorData>(content);
                    Device.BeginInvokeOnMainThread(() =>
UpdateUI(sensorData));
                }
                else
                {
                    UpdateFailureCounters();
                }
            }
            catch
            {
                UpdateFailureCounters();
            }
        }
        // Оновлення UI з новими даними з датчиків
        private void UpdateUI(SensorData sensorData)
        {
            bool temperatureReceived = sensorData.Temperature != null;
            bool humidityReceived = sensorData.Humidity != null;
            UpdateLabel(temperatureLabel, $"ТЕМПЕРАТУРА:
{sensorData.Temperature}°C", temperatureReceived);
            UpdateLabel(humidityLabel, $"ВОЛОГІСТЬ:
{sensorData.Humidity}%", humidityReceived);
            UpdateLabel(pressureLabel, $"ТИСК: {sensorData.Pressure}
hPa", sensorData.Pressure != null);
            UpdateLabel(altitudeLabel, $"ВИСОТА: {sensorData.Altitude}
m", sensorData.Altitude != null);
            UpdateLabel(timestampLabel, $"ДАТА І ЧАС ОНОВЛЕННЯ:
{sensorData.Timestamp}", sensorData.Timestamp != null);
            if (!humidityReceived)
```

```
            {
                if (++failedHumidityUpdatesCounter >= 20)
                {
                    temperatureLabel.TextColor = Color.Red;
                    humidityLabel.TextColor = Color.Red;
                    dht22_statusLabel.FormattedText =
CreateStatusText("DHT22 СТАТУС:", " Увага! Датчик DHT22 вийшов з ладу,
необхідна заміна ✗", Color.Red);
                }
            }
            else
            {
                failedHumidityUpdatesCounter = 0;
                temperatureLabel.TextColor = Color.Default;
                humidityLabel.TextColor = Color.Default;
                dht22_statusLabel.FormattedText =
CreateStatusText("DHT22 СТАТУС:", " Працездатний ☑", Color.Green);
            }
            UpdateBMP280Status(sensorData.Pressure.HasValue);
        }
        // Оновлення тексту та кольору мітки в залежності від
отриманих даних
        private void UpdateLabel(Label label, string newText, bool
dataReceived)
        {
            if (dataReceived)
            {
                label.Text = newText;
                label.TextColor = Color.Default;
            }
        }
        // Оновлення лічильників невдач, якщо дані не отримані
        private void UpdateFailureCounters()
        {
            UpdateLabel(temperatureLabel, temperatureLabel.Text,
false);
            UpdateLabel(humidityLabel, humidityLabel.Text, false);
            UpdateLabel(pressureLabel, pressureLabel.Text, false);
            UpdateLabel(altitudeLabel, altitudeLabel.Text, false);
```

```
        }
        // Оновлення статусу датчика BMP280
        private void UpdateBMP280Status(bool hasData)
        {
            if (hasData)
            {
                bmp280_statusLabel.FormattedText =
CreateStatusText("GY-BMP280-3.3 СТАТУС:", " Працездатний ☑",
Color.Green);
            }else
            {
                pressureLabel.TextColor = Color.Red;
                altitudeLabel.TextColor = Color.Red;
                bmp280_statusLabel.FormattedText =
CreateStatusText("GY-BMP280-3.3 СТАТУС:", " Увага! Ваш датчик GY-
BMP280-3.3 вийшов з ладу, необхідно замінити ✖", Color.Red);
            }
        }
        // Створення форматованого тексту для статусів датчиків
        private FormattedString CreateStatusText(string title, string
status, Color statusColor)
        {
            return new FormattedString
            {
                Spans =
                {
                    new Span { Text = title, ForegroundColor =
Color.Default },
                    new Span { Text = status, ForegroundColor =
statusColor }
                }
            };
        }
        // Метод для запам’ятовування дії про відключення від станції
        private async void OnDisconnectButtonClicked(object sender,
EventArgs e)
        {
            // Зазначаємо, що користувач більше не підключений
            Application.Current.Properties[MainPage.IsConnectedKey] =
```

```
false;
            await Application.Current.SavePropertiesAsync();
            // Навігація для повернення на головну сторінку
            await Navigation.PopAsync();
        }
        // Клас для зберігання даних з датчиків
        public class SensorData
        {
            public float? Temperature { get; set; }
            public float? Humidity { get; set; }
            public float? Pressure { get; set; }
            public float? Altitude { get; set; }
            public string Timestamp { get; set; }
        }
    }
}
```

InstructionForUser.xaml:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Full_App.InstructionForUser">
<ScrollView>
        <StackLayout VerticalOptions="FillAndExpand" Padding="30">
            <Label Text="Інструкція для користувача"
                   FontAttributes="Bold"
                   FontSize="Large"
                   HorizontalOptions="Center"/>
            <Label Text="Важливо: щоб отримувати дані з метеостанції
ваш девайс має знаходитися в одній мережі Wi-Fi з метеостанцією."
                   FontSize="16"
                   FontAttributes="Italic"/>
            <Label Text="Кроки для підключення навдено нижче"

                   FontAttributes="Bold"
                   Margin="0,10,0,10"
                   FontSize="Large"
                   HorizontalOptions="Center"/>
            <Label Text="Raspberry Pi 4"
                   FontAttributes="Bold"
```

```
                    FontSize="Large"/>
            <Label Text="1. Підключіть кабель живлення до метеостанції
Raspberry Pi 4" />
            <Label Text="2. З'єднайте вашу метеостанцію кабелем HDMI з
монітором" />
            <Label Text="3. Перейдіть у розділ Wi-Fi як показано на
фото" />
            <Frame CornerRadius="10"
                   Padding="0"
                   HasShadow="False"
                   Margin="0,10,0,10">
                <Image Source="raspberry_wifi.png"
                       HorizontalOptions="Center"
                       Aspect="AspectFit" />
            </Frame>
            <Label Text="4. Знайдіть вашу маршрутизатор в списку і
введіть пароль" />
            <Frame CornerRadius="10"
                   Padding="0"
                   HasShadow="False"
                   Margin="0,10,0,10">
                <Image Source="raspberry_wifi_connect.png"
                       HorizontalOptions="Center"
                       Aspect="AspectFit" />
            </Frame>
            <Label Text="5. Перезавантажте метеостанція і зачекайте
близько 2 хвилин доки відбудуться автоматичні налаштування
метеостанції Raspberry Pi 4" />
            <Label Text="Android"
                   FontAttributes="Bold"
                   Margin="0,20,0,0"
                   FontSize="Large"/>
            <Label Text="1. Переконайтеся що ви і ваша метеостанція і
пристрій Android підключенні до однієї мережі Wi-Fi" />
            <Frame CornerRadius="10"
                   Padding="0"
                   HasShadow="False"
                   Margin="0,10,0,10">
                <Image Source="phone_wifi.jpg"
```

```
                        HorizontalOptions="Center"
                        Aspect="AspectFit" />
            </Frame>
            <Label Text="2. Натисніть на кнопку &quot;Почати
пошук&quot; і очікуйте, ваша станція буде знайдена за лічені секунди"
/>
            <Frame CornerRadius="10"
                   Padding="0"
                   HasShadow="False"
                   Margin="0,10,0,10">
                <Image Source="phone_connect.jpg"
                        HorizontalOptions="Center"
                        Aspect="AspectFit" />
            </Frame>
            <Label Text="3. Оберіть &quot;З'єднатися&quot;, після
одразу буде отримано дані з метеостанції" />
            <Frame CornerRadius="10"
                   Padding="0"
                   HasShadow="False"
                   Margin="0,10,0,10">
                <Image Source="phone_connect_done.jpg"
                        HorizontalOptions="Center"
                        Aspect="AspectFit" />
            </Frame>
            <Label Text=" "/>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

InstructionForUser.xaml.cs

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Full_App
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class InstructionForUser : ContentPage
    {
        public InstructionForUser()
        {
```

```
            InitializeComponent();
        }
    }
}
```

Додаток И

ПРОТОКОЛ ПЕРЕВІРКИ БАКАЛАВРСЬКОГО ДИПЛОМНОГО ПРОЄКТУ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва проекту: Мікрокомп’ютерна погодна станція засобами IoT

Тип проекту: бакалаврський дипломний проєкт
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 96,8% Схожість 3,2%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор проєкту _____ Морозов П.В.
(підпис) (прізвище, ініціали)

Керівник проєкту _____ Богомолов С.В.
(підпис) (прізвище, ініціали)