Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Мікроконтролерна система моніторингу компонентів довкілля. Програмна частина
Назва теми

КвРКІ.200245.20.02.22 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-20-2 _____ В. С. Сєвостьянов
Підпис Ініціали, прізвище

Керівник _____ Д. М. Медзатий
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«19» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"10" 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Сєвостьянову Владиславу Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Мікроконтролерна система моніторингу компонентів довкілля. Програмна частина

Керівник проекту (роботи) Медзатий Д.М., к.т.н., доц.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2024 р. № 5

2. Строк подання студентом проекту (роботи) на кафедру 24.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Теоретичні основи досліджуваної проблеми

Проектування програмно-технічного засобу

Програмна реалізація та тестування програмно-технічного засобу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Блок-схеми алгоритмів роботи програмного забезпечення

Структура мікроконтролерної системи

Результати роботи програмного забезпечення

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., д.т.н., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2024	виконано
4	Робота над розділом 2 – проектування програмно-технічного засобу	01.04.2024	виконано
5	Робота над розділом 3 – програмна реалізація та тестування програмно-технічного засобу	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	30.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ В. С. Сєвостьянов
Підпис Ініціали, прізвище

Керівник роботи _____ Д. М. Медзатий
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ.200245.20.02.22 ПЗ	Пояснювальна записка	57		
			Графічні матеріали			
2		КвРКІ.200245.20.02.22 Е8	Блок-схеми алгоритмів роботи програмного забезпечення	1		
3		КвРКІ.200245.20.02.22 Е8	Структура мікроконтролерної системи	1		
4		КвРКІ.200245.20.02.22 Е8	Результати роботи програмного забезпечення	1		

КвРКІ.200245.20.02.22 ПЗ

Зм	Арк	№ докум	Підпис	Дата
Розробив		Севостьянов		19.06
Перевір.		Медзатий		19.06
Н. контр.		Лисенко		19.06
Затв.		Говорущенко		

Відомість проекту

Літера	Аркуш	Аркушів
У	1	57

ХНУ, КІ2-20-2

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мікроконтролерна система моніторингу компонентів довкілля. Програмна частина».

Автор роботи: Сєвостьянов Владислав Сергійович.

Керівник роботи: Медзатий Дмитро Миколайович.

Пояснювальна записка: 57 с., 32 рис., 3 дод., 48 джерел.

Графічна частина: 3 креслення.

МОНІТОРИНГ, БАЗА ДАНИХ, МІКРОКОНТРОЛЕР, ОДНОПЛАТНИЙ КОМП'ЮТЕР, МОДЕЛЬ ВІДОБРАЖЕННЯ.

Метою дипломної роботи є розробка програмного забезпечення мікроконтролерної системи моніторингу компонентів довкілля, його імплементація та тестування.

Об'єктом дослідження є функціональні вимоги програмного забезпечення мікроконтролерної системи моніторингу компонентів довкілля.

Предметом дослідження є програмне забезпечення мікроконтролерної системи моніторингу компонентів довкілля.

Для досягнення поставленої задачі було використано методи теоретичного аналізу та синтезу, а також метод проектування.

Розроблене програмне забезпечення має практичне застосування для роботи мікроконтролерної системи моніторингу компонентів довкілля.

20.06.2024

Підпис студента

Дата

ЗМІСТ

ВСТУП

Якість повітря є одним з найбільш важливих чинників упливу на здоров’я людей. Забруднене повітря є головною загрозою здоров’я та процвітання людей. Щороку забруднене повітря всіх форм спричиняє близько 6,5 мільйонів смертей [1]. Саме тому важливо знати, наскільки є забрудненим повітря для людини в місці, де вона проживає.

Розробка мікроконтролерної системи моніторингу компонентів довкілля є комплексною задачею, що включає в себе апаратне, програмне та серверне забезпечення.

Метою дипломної роботи є розробка програмного забезпечення мікроконтролерної системи моніторингу компонентів довкілля, його імплементація та тестування.

Програмна частина визначає функціональні можливості системи, саме тому під час дослідження мають бути визначені наперед алгоритми роботи програмного забезпечення та його функціональні вимоги.

У результаті виконаної роботи повинно бути розроблено програмне забезпечення мікроконтролерної системи моніторингу компонентів довкілля відповідно до визначених вимог.

1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Повітря є одним із найважливіших елементів навколишнього середовища та має безпосередній вплив на здоров'я людей, екосистему та клімат. Забрудненість повітря стає серйозною проблемою в багатьох містах та регіонах світу через індустріалізацію, транспорт, енергетику та інші види людської діяльності [2].

Проблема забрудненості повітря завжди була й буде актуальною, зважаючи на численні природні та людські забруднювачі, такі як: лісові пожежі, виверження вулканів, промисловість, вихлопні гази та багато іншого.

Серед наслідків забрудненого повітря можна навести наступні [3]:

– забруднене повітря містить токсичні речовини, які можуть спричиняти різноманітні захворювання дихальних шляхів, серцево-судинної системи, а також призводити до погіршення стану загального здоров'я;

– забруднення повітря може мати шкідливий вплив на рослини, тварин та водні екосистеми, що може призводити до зменшення біорізноманіття та зниження продуктивності екосистем;

– деякі забруднюючі речовини, такі як парникові гази, сприяють глобальному потеплінню та змінам клімату, що може мати серйозні наслідки для екосистем, сільського господарства та людського життя.

Серед наслідків забрудненого повітря є парниковий ефект, що спричиняє глобальне потепління. Речовинами, що викликають парниковий ефект є вуглекислий газ, метан та водяна пара – саме ці гази називають парниковими. Діаграма впливу парникових газів на клімат Землі зображено на рисунку 2.1.

Парниковий ефект діє наступним чином: частина вхідного сонячного світла відбивається атмосферою та поверхнею Землі, але більша частина поглинається поверхнею, яка нагрівається. Потім поверхня відбиває інфрачервоне (ІЧ) випромінювання. Частина ІЧ-випромінювання потрапляє в космос, але частина поглинається парниковими газами атмосфери (особливо водяною парою,

вуглекислим газом та метаном) і відбивається в усіх напрямках, частина в космос, а частина назад до поверхні, де вона ще більше нагріває Землю та нижні шари атмосфери [4].

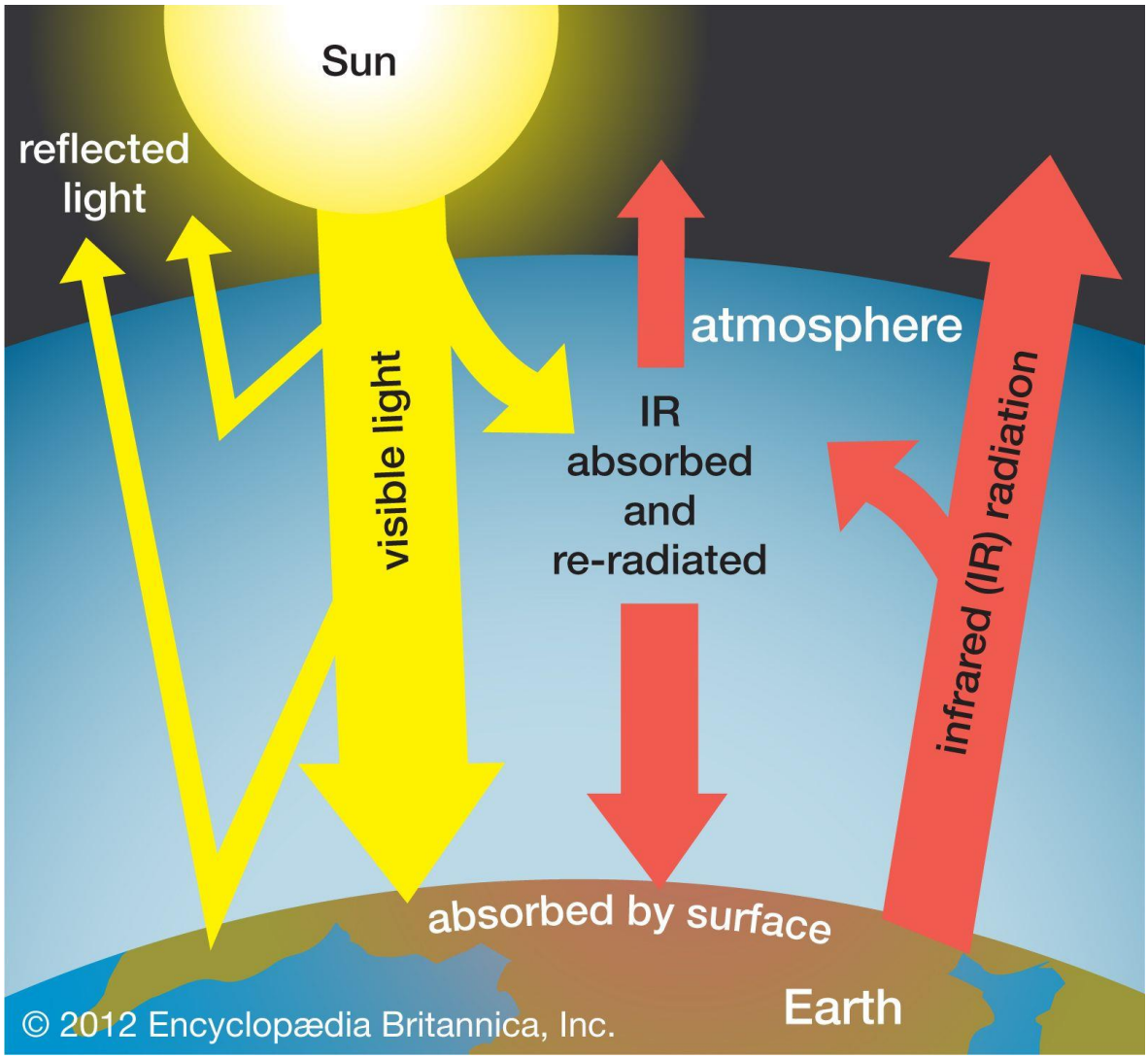


Рисунок 2.1 – Принцип дії парникового ефекту

Ще з часів початку індустріальної революції людство почало вимірювати кількість викидів вуглекислого газу в повітря.

Як можна бачити з графіку на рисунку 2.2, кількість викидів CO2 зростає експоненційно щороку [5, 6].

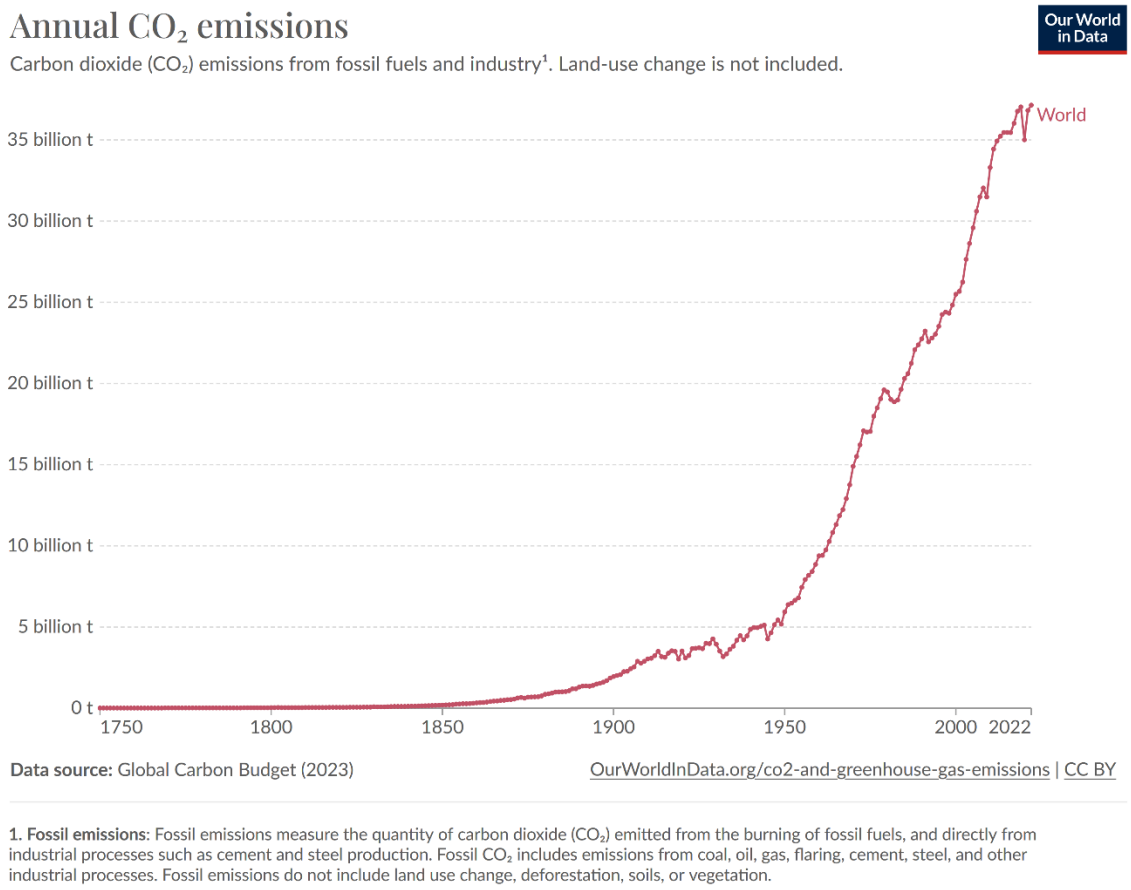


Рисунок 2.2 – Графік викидів вуглекислого газу світом в атмосферу

Історія вимірювання забрудненості повітря сягає далеко назад і пов'язана з початком промислової революції та зростанням містобудівних та промислових центрів.

З початком промислової революції у XVIII столітті в Європі та США, збільшення виробництва і використання вугілля та інших палив стало причиною значного забруднення повітря [7].

Уперше систематичне вимірювання забрудненості повітря почалося у Великі Британії в 1852 році, коли Шотландський хімік на ім’я Роберт Енгус Сміт виявив кислотний дощ після того, як зібрав зразки дощу та дослідив їх. У цьому дослідженні Сміт дізнався, що краплини лощу містять велику кількість сірки через спалювання вугілля. Таким чином, Енгус Сміт став першим вченим, що провів дослідження хімічної кліматології забрудненої атмосфери [8].

Усе ж, першим пристроєм для виміру забруднення повітря став так званий датчик відкладень (англ. Deposit gauge) (рис. 2.3) – по аналогії до опадоміру, що збирає дощову воду для виміру кількості опадів, датчик відкладень вимірював вміст сажі у повітрі. Уперше датчик відкладень був розроблений у ранньому 20-у сторіччі, дизайн якого в подальшому було удосконалено та стандартизовано Комітетом Дослідження Атмосферного Забруднення (англ. Committee for the Investigation of Atmospheric Pollution) [9].

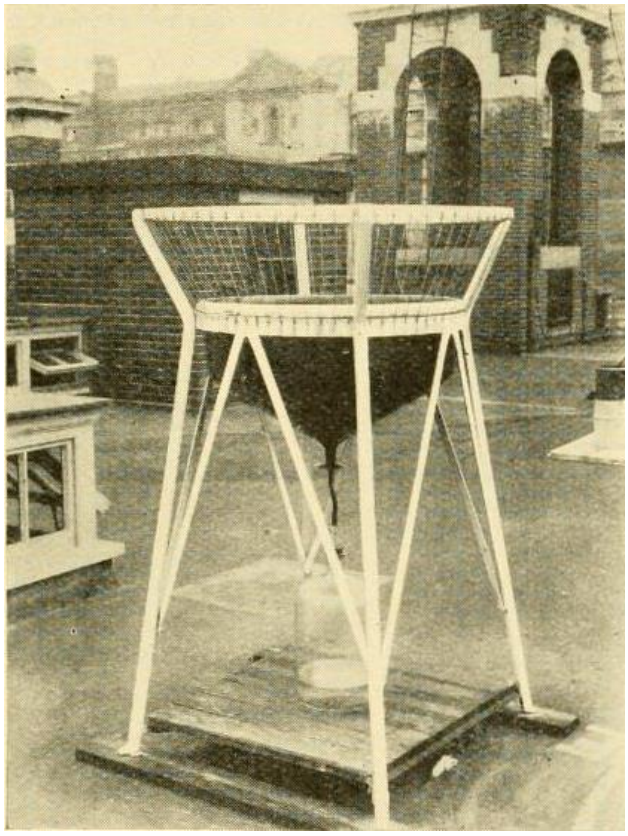


Рисунок 2.3 – Рання модель датчика відкладень в 1925 році

У середині XIX століття Лондон зіткнувся із загрозливою екологічною кризою, яку прозвали «Великий смог» [4]. Швидкий триб індустріалізації побудив переходити місто на викопне паливо, що побудило стрімке накопичення шкідливих речовин у повітрі. Це поєднання вугільного диму, сажі та хімічних викидів призвело до утворення щільної хмари смогу, яка огорнула місто. Це мало руйнівний вплив на здоров’я мешканців: повітря стало непридатним для дихання.

Астма, респіраторні захворювання та рання смерть стали поширеними, що викликало тривогу серед населення. За деякими підрахунками в результаті цієї катастрофи загинуло близько 12000 людей [10]. На рисунку 2.4 наведено графік, що показує кореляцію між вмістом діоксиду сірки та кількістю смертей в Лондоні в період Великого смогу [11].

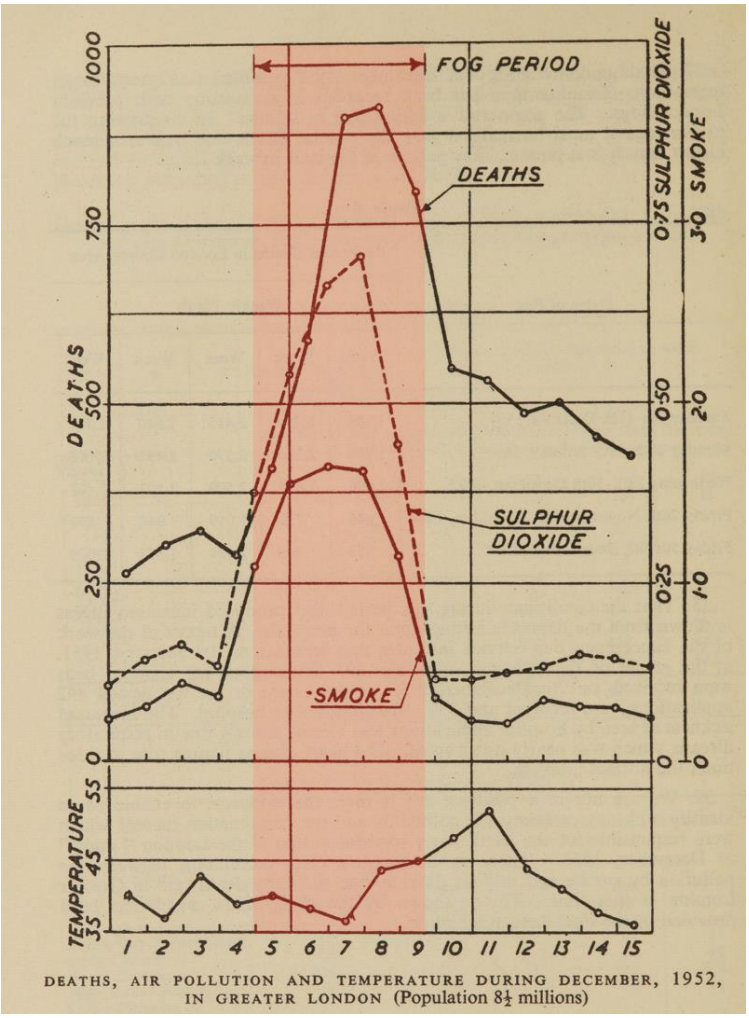


Рисунок 2.4 – Графік кількості смертей та вмісту діоксиду сірки в Лондоні в грудні 1952

Через посилення смогу уряд та громадськість були все більш занепокоєні, що призвело до рішучих дій та запровадження у 1956 році Акту Про Чисте Повітря (англ. Clean Air Act) [12]. Це акт мав на меті зменшення кількості спалення вугілля та сприяв переходу на більш чисті джерела енергії.

Ця подія назавжди стала трагічним нагадуванням про наслідки забруднення повітря не тільки для жителів Лондону, а й для всього світу.

У XX столітті були розроблені більш точні та чутливі методи аналізу, які дозволили вимірювати концентрації різних хімічних речовин у повітрі. Це дозволило науковцям отримувати більш детальну інформацію про склад та забруднення атмосфери.

З розвитком технологій та зростанням обізнаності про екологічні проблеми, були створені мережі моніторингу якості повітря у багатьох країнах світу. Ці мережі використовуються для збору даних про рівні забруднення у різних місцях світу.

Сьогоднішній розвиток сенсорів, дронів, супутникових технологій та інших інноваційних методів дозволяє отримувати більш точні та повних дані про забруднення повітря. Це допомагає виявляти забруднення, визначати їхні джерела та розробляти стратегії для його зменшення.

Узагальнюючи, вимірювання забруднення повітря є важливою складовою екологічного моніторингу, яка розвивалася протягом століть і продовжує розвиватися зараз у зв'язку з ростом свідомості про проблеми забруднення довкілля та застосуванням сучасних технологій.

Хоч забруднення вуглекислим газом є шкідливим для здоров’я людей та клімату, усе ж це не є єдиним забруднювачем. Також серед типів забруднення існують наступні [2]:

– тверді частки, що включають пил, дим, дрібні частки, які можуть вдихатися та потрапляти в легені, спричиняючи захворювання;

– газові забруднення: оксиди азоту, оксиди сірки, озон та інші гази можуть бути токсичними та шкідливими для здоров'я;

– викиди вуглеводнів та інших хімічних сполук: викиди від автомобілів, промислових підприємств та інших джерел можуть містити хімічні речовини, які є отруйними та небезпечними для здоров'я;

– забруднення озоном: озон виробляється природнім чином у верхніх шарах атмосфери, коли сонячне світло взаємодіє із киснем. Також озон може вироблятися через спалювання викопного палива, від авто з двигуном внутрішнього згоряння, через роботу фабрик та об’єктів промисловості.

Забруднене повітря може мати незначний вплив, як подразнення очей, слизової, що може викликати кашель, чхання, якщо концентрація пилу не висока, але при високій концентрації забруднене повітря може викликати респіраторні захворювання, астму, хронічні захворювання пов’язані з дихальною системою та навіть рак [3]. Найбільший плив на здоров’я мають люди в групі ризику (діти, дорослі віком від 65 років, люди із хворобами легень, астмою та серцевими хворобами, вагітні). За даними Національного Інституту Здоров’я (NIH) забруднене повітря провокує близько 6,5 мільйонів смертей щорічно в усьому світі [1]. Тому так важливо знати ступінь забрудненості повітря в локальній місцевості.

Розрізняють два типи вимірювань забрудненості повітря: пасивне та активне [13]. Пасивне вимірювання включає в себе недорогі та прості пристрої, що збирають зразки з навколишнього повітря, поглинаючи їх. Одним із найпоширеніших пристроїв пасивного вимірювання є дифузійна трубка, яку вішають в місці, де потрібно зібрати зразки одного або декількох забруднюючих газів. Через деякий час таку трубку знімають, а її вміст досліджують у лабораторії.

Активне вимірювання включає в себе датчики, що автоматично збирають зразки за допомогою вентиляторів, або спеціальних фільтрів. Такі пристрої є автоматичними, або напівавтоматичними, що дозволяє збирати зразки з повітря та одразу отримувати результат про вміст певних речовин.

Існують різні способи вимірювання забрудненості повітря, серед них можна вирізнити 3 основні типи:

– окремі датчики повітря: використання спеціальних електронних датчиків, які вимірюють концентрацію різних забруднюючих речовин у повітрі. Це можуть бути як датчики озону, так і окремі датчики вмісту вуглекислого газу, чи пилу;

– моніторингові станції: розташування спеціалізованих моніторингових станцій у різних частинах міста або регіону для систематичного збору даних про якість повітря. Такі станції обладнані одразу багатьма датчиками для вимірювання різних типів забрудненості повітря та можуть надсилати дані через мережу інтернет;

– супутникові спостереження: використання супутникових технологій для вимірювання концентрації забруднюючих речовин у повітрі на великих територіях.

Слідкуючи за забрудненістю повітря, вимірюють наступні параметри [14]:

– кількість часток пилу: пил є різними твердими та рідкими мікроскопічними сполуками, що знаходяться в повітрі. Розрізняють кілька видів забруднення пилом, а саме: PM1 – частинки, що менші за один мікрон у діаметрі, PM2.5 – дрібні частинки, що менші за 2,5 мікрона в діаметрі, PM10 – частинки, менші за 10 мікрон у діаметрі;

– вміст формальдегідів (CH2O), вуглекислого газу (CO2), озону (O3), CO, SO2, NO2 та метану (CH4), адже саме ці сполуки є основними забруднювачами повітря;

– вміст летких органічних сполук (VOC - volatile organic compounds). Це широкий спектр органічних сполук, що включає вуглеводні, альдегіди, спирти, кетони, терпеноїди та інші;

– радіація: хоч це й не є поширеним, та все ж деякі прилади вимірюють вміст радіоактивних часток;

– температура, вологість та рівень ультрафіолетового випромінювання – хоч ці параметри не є обов’язковими, та все ж зазвичай їх також вимірюють.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

Сучасні системи моніторингу якості повітря вимірюють навколишні параметри, включаючи PM1, PM2.5, PM10, CO, CO2, SO2, NO2, O3 та H2S, а також вологість, температуру та ультрафіолетове випромінювання.

На ринку представлено чимало пристроїв вимірювання якості повітря навколишнього середовища. Усі вони відрізняються насамперед ціною та кількістю показників, що вимірюються. До прикладу можна взяти систему виміру якості повітря від компанії Oizom, яка спеціалізується на виробництві систем моніторингу якості повітря для використання у містах, на підприємствах, будівництвах тощо. Однією із систем моніторингу якості повітря є пристрій Polludrone (рис. 2.5) [15].

Рисунок 2.5 – Пристрій виміру якості повітря Polludrone від компанії Oizom

Такий пристрій здатен визначати вміст частинок PM1, PM2.5, PM10 та PM100; вміст вуглекислого газу та карбону монооксиду; вміст SO2, NO2, NO, H2S, та навіть рівень шуму. Також Polludrone обладнаний мережевими адаптерами для під'єднання до інтернету через 3G, WiFi, LORA, LTE, або Ethernet. Пристрій здатен самостійно відправляти дані до хмари компанії Oizom. Також пристрій не потребує підзарядки, адже він обладнаний сонячними панелями.

Серед переваг такого пристрою є широкий спектр вимірюваних параметрів та висока надійність, адже Polludrone має рівень захисту IP66, що означає, що такий пристрій зможе справно функціонувати майже в будь-яких погодних умовах.

Серед недоліків такого пристрою можна зазначити його великі розміри, що становлять 36 см х 33 см х 20 см, велику вагу в 7,2 кг та високу ціну, що становить від $6000, у залежності від комплектації.

Серед пристроїв вимірювання якості повітря індустріального типу можна навести систему AQBot від тої ж компанії Oizom (рисунок 2.6) [16].

Рисунок 2.6 – Індустріальний пристрій виміру якості повітря від компанії Oizom

AQBot має різні варіації: у залежності від потреб замовника, пристрій може вимірювати окремо вміст NH3, H2S, CH4, CO, PM та інші.

Серед переваг даного пристрою є його компактність, вбудований дисплей для перегляду зчитуваних параметрів, підтримка всіх стандартних з’єднань для дротового з’єднання та аналітичне програмне забезпечення для перегляду даних, їх аналізу та генерування звітів та попереджень про забрудненість повітря на основі зібраних даних. Також AQBot має вбудовану систему сповіщень, щоб додатково повідомляти про ризики недопустимого вмісту забрудників у повітрі.

Серед недоліків можна заначити вузьку спеціалізацію пристрою.

Розглянемо пристрої вимірювання якості повітря для домашнього застосування. Такі монітори зазвичай є більш дешевими та не потребують технічних знань для їх встановлення, тобто вони працюють «з коробки». Одним з таких пристроїв є M10 від компанії Temtop (рис. 2.7) [17].

Рисунок 2.7 – Пристрій для моніторингу якості повітря від компанії Temtop

M10 має датчики для вимірювання вмісту PM2.5, формальдегідів та летючих органічних сполук. Також він обладнаний батареєю та дисплеєм, що підходить для використання в офісах, жилих кімнатах, чи навіть в авто. Ціна на такий пристрій становить близько 3400 грн, що є відносно невисокою ціною, у порівнянні із конкурентами.

Серед недоліків такого пристрою є невелика вибірка вимірюваних параметрів та відсутність можливості відправлення даних на сервер.

Ще одним переносним пристроєм вимірювання якості повітря є M2000C від компанії Temtop (рис. 2.8).

Рисунок 2.8 – Пристрій вимірювання якості повітря M2000C від компанії Temtop

Даний прилад може вимірювати вміст вуглекислого газу, PM2.5 та PM10, температуру, вологість та атмосферний тиск. M2000C має вбудований екран та батарею, що робить його зручним пристроєм для носіння із собою [18]. Ціна такого пристрою становить близько 6000 грн.

Серед переваг такого пристрою є невеликі розміри та зручність використання. Недоліками пристрою є невеликий набір параметрів для вимірювання, доволі висока ціна та відсутність можливості відправки зібраних даних на сервер.

Серед «розумних» пристроїв для моніторингу якості повітря можна розглянути систему uHoo Smart Air Monitor (рис. 2.9).

Рисунок 2.9 – Пристрій вимірювання якості повітря uHoo Smart Air Monitor

Даний пристрій може вимірювати показники температури, вологості, тиск, вміст PM2.5, NO2, CO, CO2, O3, вміст летючих органічних сполук [19]. Хоч uHoo Smart Air Monitor не має вбудованого монітору, він може автоматично відправляти дані на сервер, звідки вони можуть бути зчитаними в зручному форматі на мобільних пристроях з операційною системою Android, чи IOS, а також на

комп’ютері. Дане програмне забезпечення має безкоштовну та платну версії (з підпискою $9,99 на місяць). Ціна одного uHoo Smart Air Monitor становить $299.

Недоліками такого рішення є залежність пристрою від мережі інтернет, адже uHoo Smart Air Monitor не має вбудованого екрану для перегляду зчитаних параметрів у реальному часі. Також, висока ціна на пристрій є значним недоліком для подібних систем.

Отже, порівнюючи різні пристрої для вимірювання якості повітря, можна дійти висновку, що кожен з них має різні параметри вимірювання, у залежності від сфери застосування. Також однією з відмінних рис є можливість надсилання зчитаних даних через інтернет для подальшого їх відображення в спеціальних застосунка. Основним недоліком даних систем є висока ціна та невелика вибірка параметрів вимірювання якості повітря у відносно не дорогих моделях.

1.3 Методологічні підходи до вирішення задачі за темою дослідження

Стенд моніторингу за якістю повітря із можливістю відправлення даних через мережу інтернет на сервер має складатися із датчиків, які будуть вимірювати певні показники та мікрокомп’ютера, що буде ці показники оброблювати та надсилати на сервер. Такий відхід забезпечить компактність та переносимість самого пристрою.

Для написання програмного забезпечення для пристрою доцільно використовувати мови програмування, що найбільше використовуються, та для яких пишуть відповідні бібліотеки розробники датчиків. Такими мовами можуть бути Python, чи MicroPython.

Для створення застосунку для смартфону можна використати мови програмування як JavaScript, Kotlin, Java, або Swift, у залежності від обраного фреймворку та цільової платформи.

1.4 Постановка задачі

Задачею роботи є створення програмного забезпечення для стенду моніторингу компонентів довкілля.

Першим етапом є формування об’єкту та мети поставленої задачі, що включає теоретичне вивчення тем, пов’язаних із розробкою стенду моніторингу компонентів повітря.

Другим етапом є дослідження функціональних особливостей програмного забезпечення системи моніторингу компонентів повітря та визначення основних алгоритмів роботи.

Третім етапом є проведення теоретичного аналізу сфери програмування систем моніторингу компонентів довкілля, що включає вибір мов програмування та середовищ розробки програмного забезпечення.

Четвертим етапом є визначення предметної області та її структури.

П’ятим етапом є визначення функціональних вимоги розроблювального програмного забезпечення для пристрою моніторингу компонентів довкілля.

Заключним етапом є безпосередньо розробка програмного забезпечення для системи моніторингу компонентів довкілля.

1.5 Висновки

Беручи до уваги наявні рішення, їх функціональність та можливості, буде спроектовано та побудовано мікроконтролерну систему моніторингу компонентів довкілля. Поставлена задача повинна буде відповідати визначеним функціональним вимогам, а також повинна бути детально досліджена.

Програмна імплементація проаналізованого рішення повинна бути оцінена та описана алгоритмічно, враховуючи можливості вибраного апаратного забезпечення.

Розробка програмного забезпечення повинна відбуватися в чотири етапи для забезпечення розробки найбільш ефективного рішення:

– розробка алгоритму виконуваного програмного забезпечення;

– імплементація програмного забезпечення відповідно до розробленого алгоритму;

– тестування та розробленого програмного забезпечення;

– внесення змін в реалізацію за потреби.

Отже, розроблювальне програмне забезпечення повинне відповідати всім поставленим вимогам, бути чітко сформульованим та алгоритмічно описаним.

2 ПРОЕКТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

2.1 Обґрунтування вибору мов програмування та програмного забезпечення

Серед пристроїв моніторингу компонентів довкілля домінантною мовою програмування є Python, якщо використовувати мікроконтролери Raspberry Pi. Серед мікроконтролерів Arduino також використовується мова C. Оскільки для реалізації проєкту було взято за основу одноплатний комп'ютер Raspberry Pi, тому основною мовою програмування було обрано мову Python.

Мова програмування Python є потужною та простою у вивченні мовою, що має повний функціонал об'єктно-орієнтованої мови програмування, при цьому не потребує знань операцій низькорівневих мов, як от роботи з адресами пам'яті та вивільненні їх вручну [20]. Це все дозволяє швидше розробляти потрібний функціонал програм, не продумуючи логіки низькорівневого програмування, адже це все автоматизовано на рівні мови програмування. Також для Python створено тисячі бібліотек для різних задач та потреб [21]. Не виключенням є й модулі моніторингу якості повітря. Саме тому було обрано мову програмування Python.

Середовищем програмування мікроконтролера Raspberry Pico було обрано програмне забезпечення Thonny, що має вигляд, зображений на рисунку 2.1.

Thonny поставляється з передвстановленим інтерпретатором Python та підтримує компілятор MicroPython.

Основною перевагою та причиною вибору даного середовища розробки є можливість з'єднання з мікроконтролером Raspberry Pico у реальному часі та швидке завантаження коду на плату без додаткових маніпуляцій та втручання.

Також Thonny містить функцію автоматичної інсталяції компілятора MicroPython на обраний мікроконтролер, що значено спрощує роботу, адже підготовка мікроконтролера не потребує ручного встановлення потрібного компілятора.

Оскільки деякі датчики як датчик пилу GP2Y1010AU0F та датчик витоку газу MQ-135 передають дані аналоговим сигналом, було додано до схеми мікроконтролер Raspberry Pico, який має у собі кілька аналогових входів.

Саме тому з'явилася потреба в написанні частини коду, що отримує та частково оброблює сигнал з датчиків, що буде виконуватися за допомогою компілятора MicroPython.

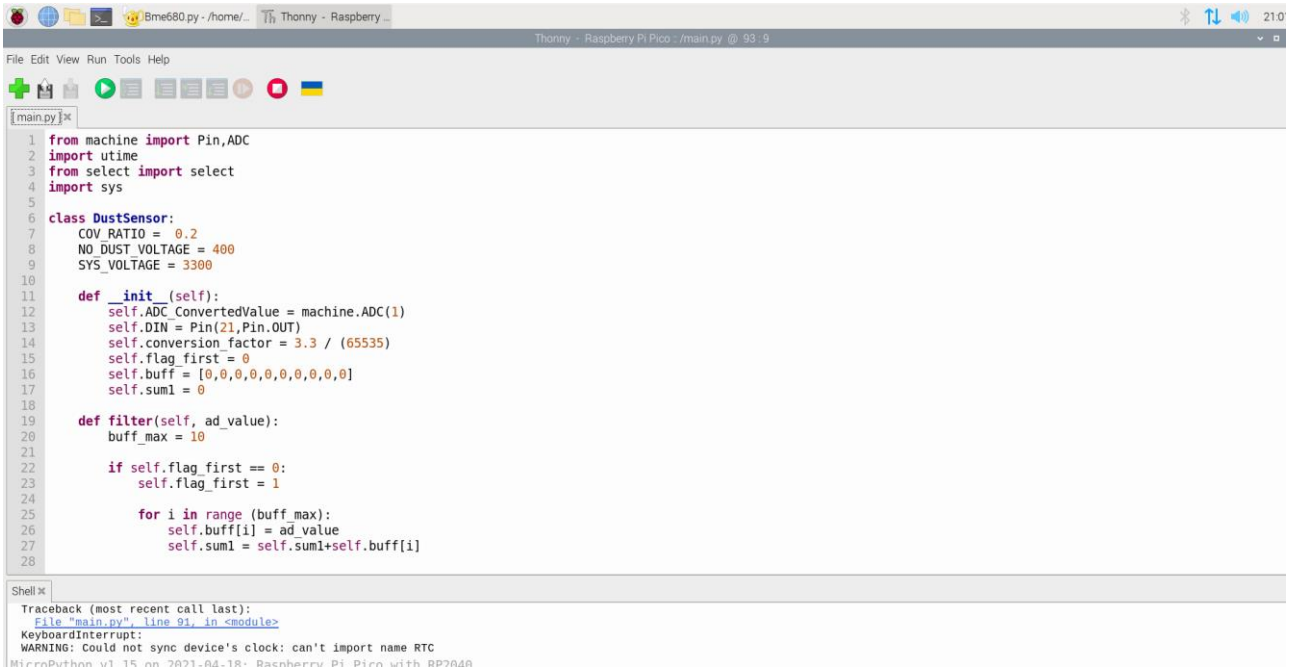


Рисунок 2.1 — Графічний інтерфейс середовища розробки програмного забезпечення Thonny

Для програмування одноплатного комп'ютера Raspberry Pi було обрано передвстановлений редактор коду Geany, що є безкоштовним редактором коду з деяким функціоналом інтегрованого середовища розробки програмного забезпечення.

Geany є дуже легким у навантаженні та простим у використанні. На рисунку 2.2 зображено графічний інтерфейс Geany.

Для створення Android застосунку було обрано мову програмування Kotlin. Дана мова програмування є високорівневою багатоплатформною мовою програмування із сильною типізацією.

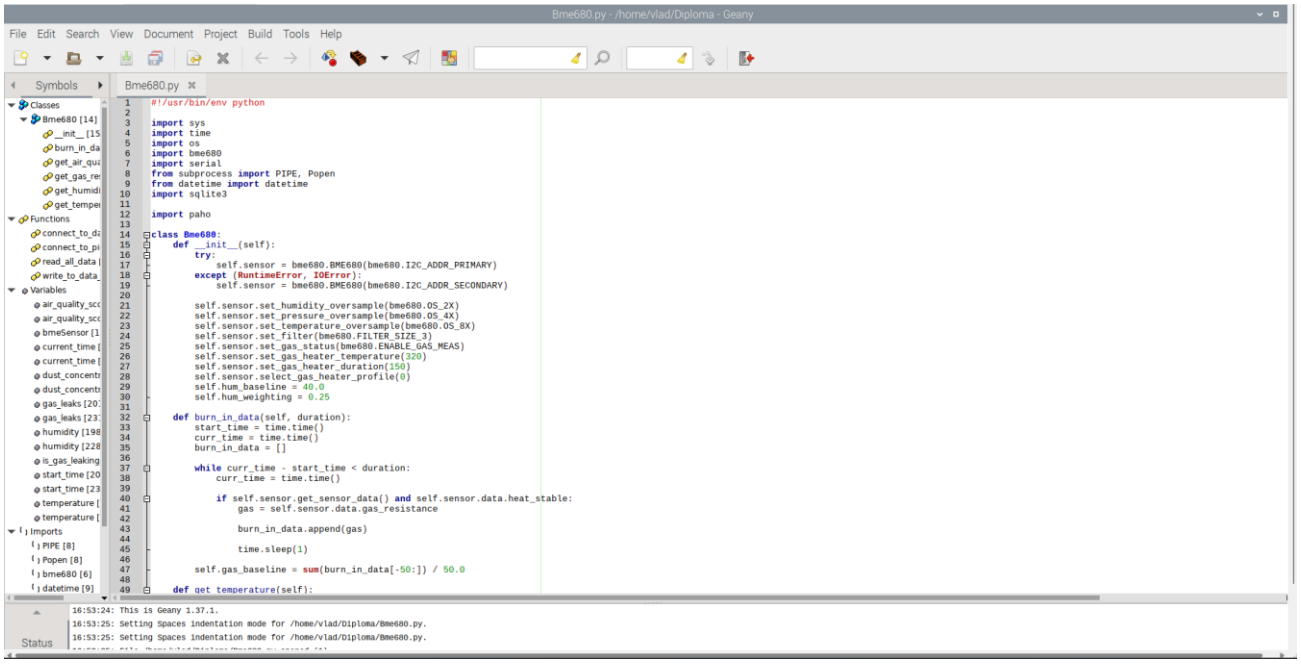


Рисунок 2.2 — Інтерфейс редактора коду Geany

Kotlin повністю інтерпретується в мову Java, тому має всі її особливості, але при цьому деякі недоліки було прибрано та загалом покращено синтаксис для швидшого та ефективного програмування [22]. Серед покращень можна навести відсутність у мові Kotlin перевірених винятків (англ. checked exceptions), що загалом підвищує продуктивність коду.

Мова Kotlin є сучасною мовою програмування, що має багато особливостей, які полегшують розробку програмного забезпечення та роблять код більш безпечним.

Серед особливостей мови програмування Kotlin можна навести наступні [23]:

– код, що написаний мовою Kotlin, має менший обсяг ніж той, що написаний мовою Java, маючи при цьому функціонально те ж значення;

– код мови Kotlin є більш безпечним, адже дана мова підтримує нулевість, що означає, що код матиме набагато меншу ймовірність викинути виняток посилання на NULL;

– мова Kotlin підтримує багатопотоковість та паралелізм, що дозволяє легко працювати з мережевими запитами та розподіляти навантаження між кількома потоками.

Для розробки Android застосунку мовою Kotlin було обрано інтегроване середовище розробки програмного забезпечення (IDE) Android Studio. Дане середовище є спеціально розробленим для створення програмного забезпечення для операційної системи Android. Android Studio має вбудовану підтримку мов Kotlin та Java, а також містить в собі функціонал для емуляції android-пристроїв та налагодження коду [24]. На рисунку 2.3 зображено вигляд графічного інтерфейсу користувача Android Studio.

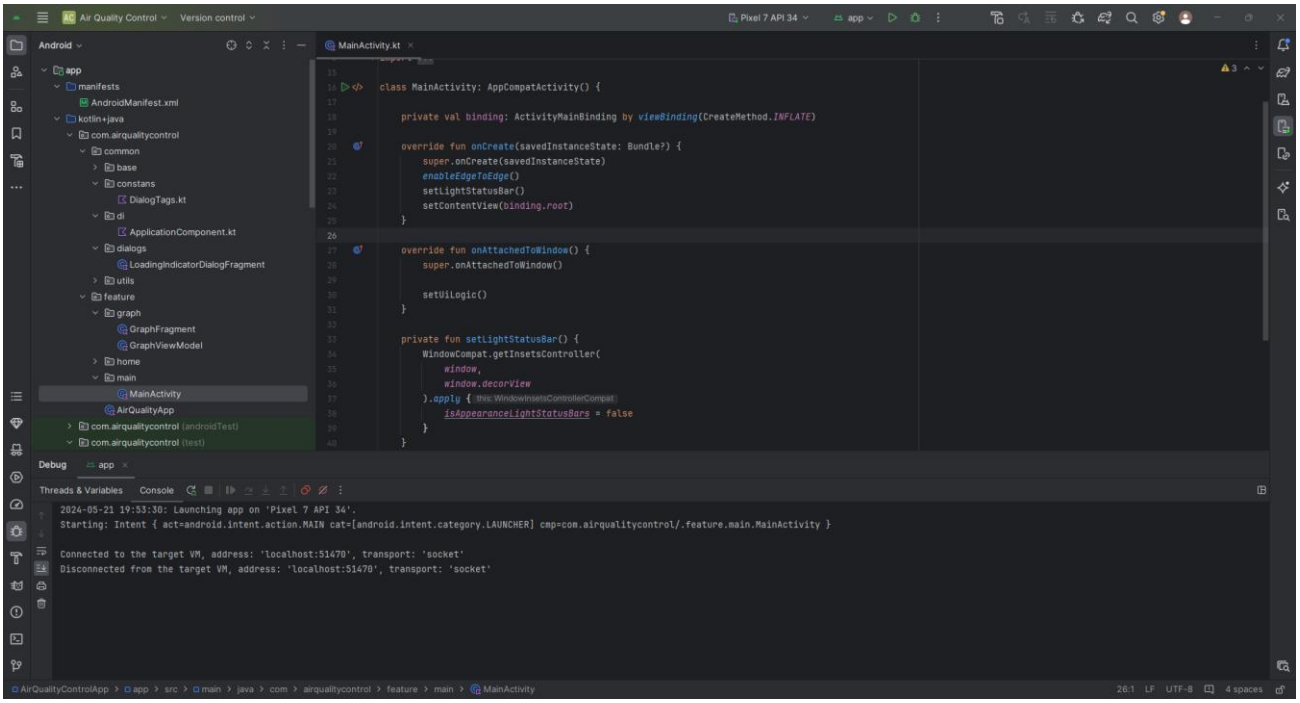


Рисунок 2.3 — Вигляд графічного інтерфейсу користувача Android Studio версії Iguana 2023.2.1

Також, як і майже всі сучасні середовища розробки програмного забезпечення, Android Studio підтримує системи контролю версій Git та Mercurial. Ще однією особливістю Android Studio є підтримка завантажуваних плагінів, що дозволяють розширити функціонал середовища та покращити процес розробки програмного забезпечення.

Для зберігання даних у пам’яті мікроконтролера було використано бібліотеку для побудови реляційної бази даних SQLite. SQLite є невеликою, швидкою, надійною та повнофункціональною бібліотекою для роботи з базою даних рушія

SQL. Оскільки мікроконтролерна система є малопотужною та має обмежену кількість пам’яті, даний вибір бібліотеки для роботи з базою даних є найкращим, адже SQLite важить менше одного мегабайта та підтримується багатьма мовами програмування, включно з мовою Python, та різними архітектурами процесорів. Також варто зазначити, що SQLite є в середньому на третину швидшою, ніж використання звичайної файлової системи [25].

Попри значні переваги в об’ємі та швидкості читання та запису, SQLite має недолік обмеженої кількості типів даних, які зберігаються в базі даних. Типи даних, які підтримуються бібліотекою SQLite наведено нижче:

– NULL – тип даних, що позначає нуль;

– INTEGER – цілочисельні дані, що зберігаються в розмірі від 0 до 8 байт;

– REAL – числа з плаваючою комою розміром 8 байт;

– TEXT – рядок тексту, що зберігається в кодуванні UTF-8, UTF-16BE або UTF-16LE;

– BLOB – тип даних, що зберігається так само, як його було введено.

Враховуючи нескладну структуру даних, що будуть зберігатися в базі даних недолік обмеженої кількості типів даних не є релевантним.

Для безпосередньої роботи з базою даних SQLite на одноплатному комп’ютері Raspberry Pi було обрано програму DB Browser for SQLite. Дане програмне забезпечення є легким в навантаженні та простим у застосуванні інструментом для створення та керування баз даних бібліотеки SQLite. Дане програмне забезпечення було обрано, адже одноплатний комп’ютер Raspberry PI має обмежену потужність процесора та невелику кількість пам’яті, через що DB Browser for SQLite є одним із найкращих варіантів для роботи з базою даних SQLite.

На рисунку 2.4 зображено вигляд програмного забезпечення DB Browser for SQLite.

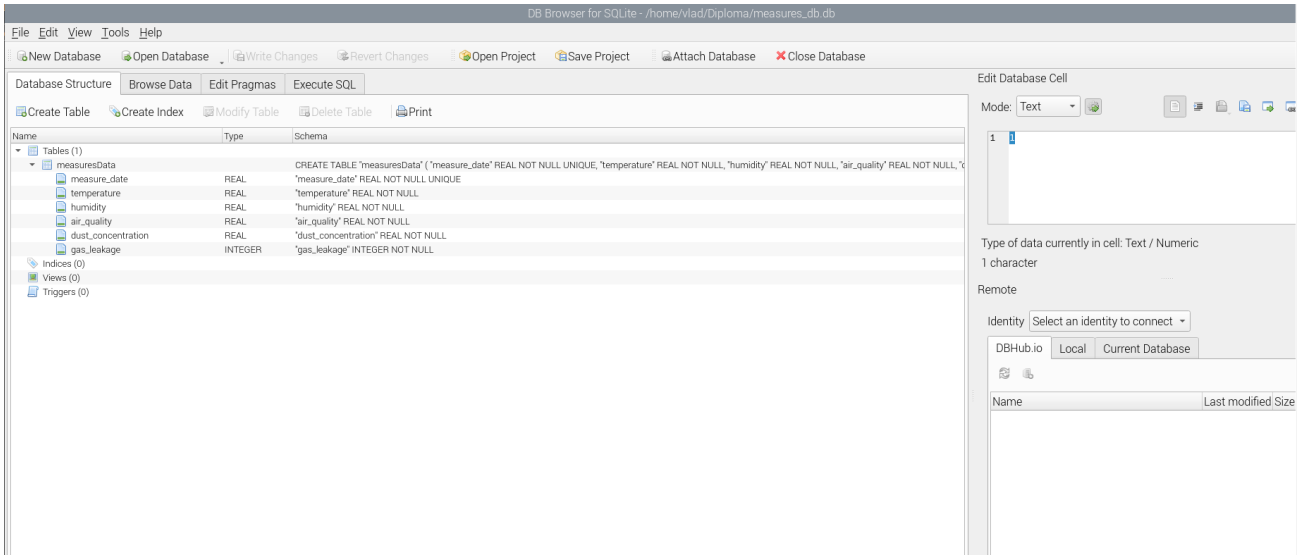


Рисунок 2.4 — Вигляд користувацького інтерфейсу програми DB Browser for SQLite

DB Browser for SQLite є безкоштовним програмним забезпеченням із відкритим кодом, що дозволяє виконувати наступні функції над базою даних SQLite за допомогою графічного інтерфейсу:

- створення та збереження файлів бази даних;
- створення, визначення та редагування таблиць;
- створення, визначення та видалення індексів;
- перегляд, редагування, додавання та видалення записів бази даних;
- сортування та пошук записів;
- імпорт та експорт записів у текстовому форматі;
- створення та виконання SQL-запитів;
- ведення журналу команд SQL-запитів;
- побудова графічної структури таблиць бази даних.

2.2Функційні вимоги програмного забезпечення

Для правильної роботи програмного забезпечення перед безпосередньою розробкою потрібно визначити його основний функціонал.

Пристрій моніторингу компонентів довкілля функційно має виконувати наступні вимоги:

– зчитування параметрів з датчиків виміру компонентів довкілля;
– збереження даних про виміри локально (кешування даних);
– відправлення зібраних даних на сервер.

Відповідно до цих функціональних вимог, можна побудувати загальну структуру роботи проєкту (рис. 2.5).

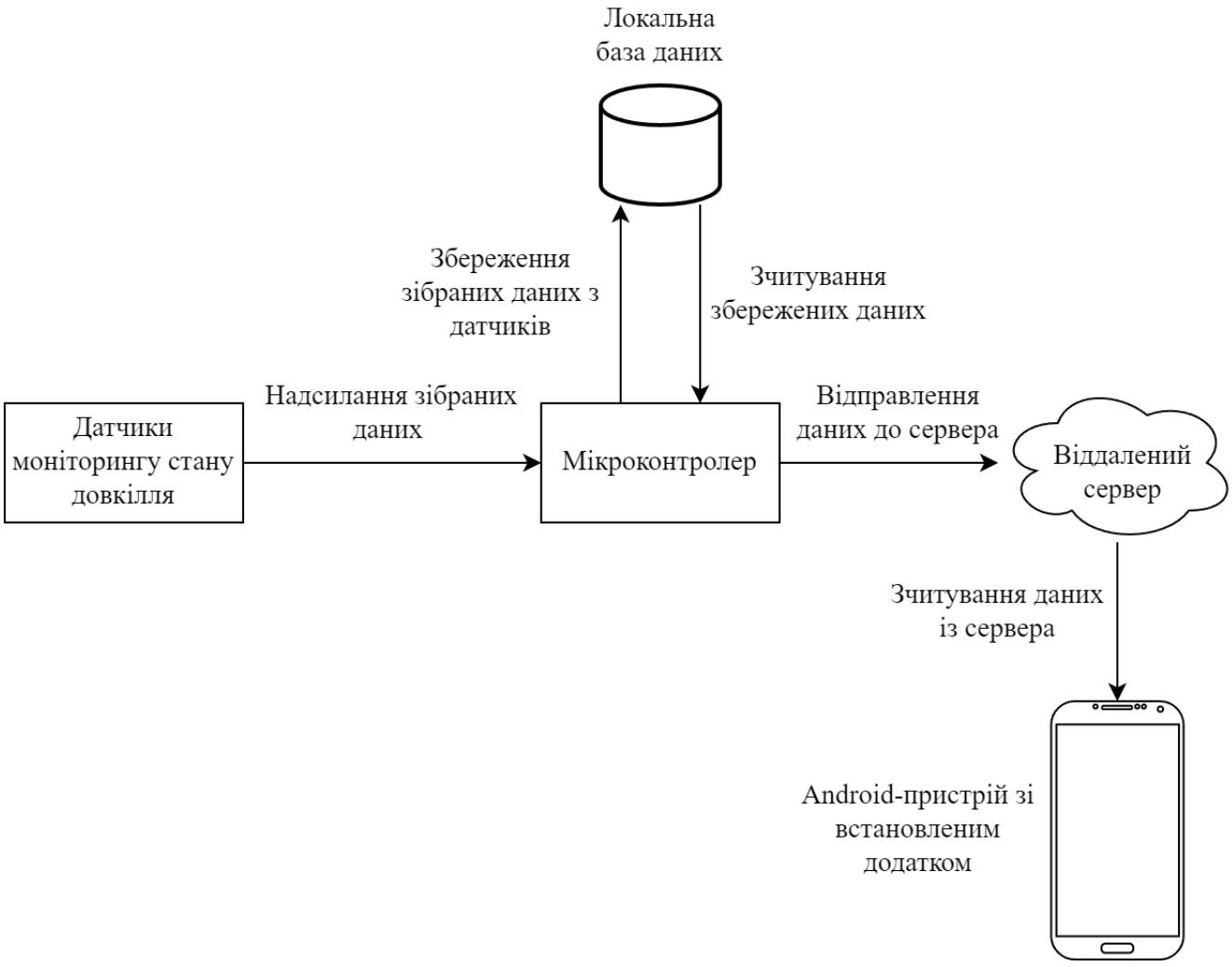


Рисунок 2.5 — Функціональна структура проєкту

Відповідно до цих вимог, програмне забезпечення мікроконтролера має зчитувати зібрані дані з датчиків, зберігати їх у локальній базі даних та періодично відправляти дані до віддаленого сервера.

У свою чергу Android застосунок має отримувати дані з сервера для їх подальшого відображення користувачу.

Відповідно до стандартизованої моделі Всесвітнього форуму Інтернету речей (IoTWF), модель якої зображено на рисунку 2.6 [26]. Можна відповідно розподілити структурні частини проєкту до певних функційних вимог.

Рівень 7 - співпраця та процеси: залучення людей та бізнес
Рівень 6 - застосунок: звітність, аналіз, контроль
Рівень 5 - абстракція даних: агрегація та доступ
Рівень 4 - збереження даних
Рівень 3 - крайові обчислення: аналіз та трансформація даних
Рівень 2 - з'єднання: пристрої зв'язку та обробки
Рівень 1 - фізичні пристрої та контролери

Рисунок 2.6 — Стандартизована модель Інтернету речей

До першого рівня IoTWF моделі можна віднести датчики моніторингу стану довкілля, що складаються з датчику пилу Waveshare Dust Sensor, що побудований на основі оптичного датчику пилу GP2Y1010AU0F [27]; датчику газу, вологості, температури та якості повітря BME 680 та датчику витоку газу MQ-135. Усі сенсори відповідають лише за безпосередній збір даних з навколишнього середовища та відправлення отриманих результатів через дротове з’єднання з мікроконтролерами.

Мікроконтролер Raspberry Pico та одноплатний комп’ютер Raspberry Pi можна віднести до другого та третього рівнів моделі. Саме ці пристрої

забезпечують з’єднання із датчиками та віддаленим сервером, а також виконують часткову обробку даних, декодуючи отримані результати із сенсорів.

До рівнів 4 та 5 можна віднести віддалений сервер, що одночасно виконує збереження отриманих даних з датчиків за протоколом MQTT та надання доступу до них за допомогою протоколу HTTP. Протокол MQTT було обрано через його особливість роботи із з’єднанням з малою пропускною здатністю [28]. Протокол HTTP є надійним рішенням для надання доступу до даних кінцевому користувачу через Android застосунок.

Розроблений застосунок можна віднести до рівнів 6 та 7, адже саме тут відбувається презентація даних у форматі, зрозумілому для людини. Програмне забезпечення застосунку для ОС Android повинне отримувати дані з віддаленого сервера за протоколом HTTP та виводити на екран у доступному форматі [29].

2.3 Проектування роботи програмного забезпечення мікроконтролера

Мікроконтролер є керівною частиною системи моніторингу якості повітря на крайовому рівні, з чого випливають основні функції програмного забезпечення мікроконтролера [30].

Відповідно до вимог мікроконтролера, має відбуватися збір даних з датчиків, їх збереження в локальній базі даних та відправлення готових даних до віддаленого серверу. Звідси можна побудувати схему основних компонентів програмного забезпечення мікроконтролера (рис 2.7).

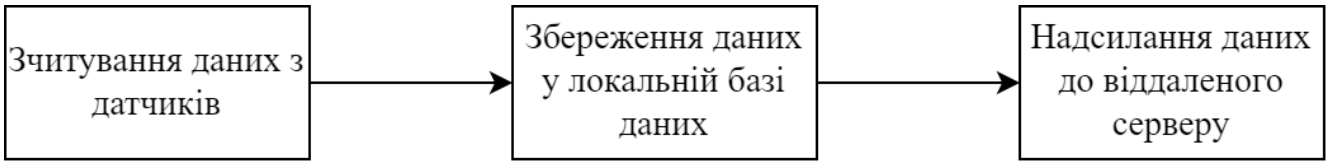


Рисунок 2.7 — Компоненти програмного забезпечення мікроконтролера

Зчитування даних з датчиків має супроводжуватися їх декодуванням та перетворенням до зрозумілих для людини величин [31]. Функцією цього

компонента є безпосереднє зчитування сигналів з датчиків. Після отримання даних необхідно їх конвертувати у відповідні величини: для температури — градуси Цельсія, для вологості повітря — відсотки, для концентрації частинок пилу — мікрограми на метр кубічний, для показника якості повітря — відсотки, для наявності викиду газу — булеве значення та для тиску — гектопаскалі [32]. Для деяких показників, як концентрація частинок та показник якості повітря, необхідно виконати попереднє калібрування сенсорів та фільтрацію отриманих значень [33]. Показник якості повітря обчислюється з опору газу: чим більший опір, тим чистіше повітря; та з вологості повітря. Блок-схема роботи програмного забезпечення мікроконтролера зображено на рисунку 2.8.

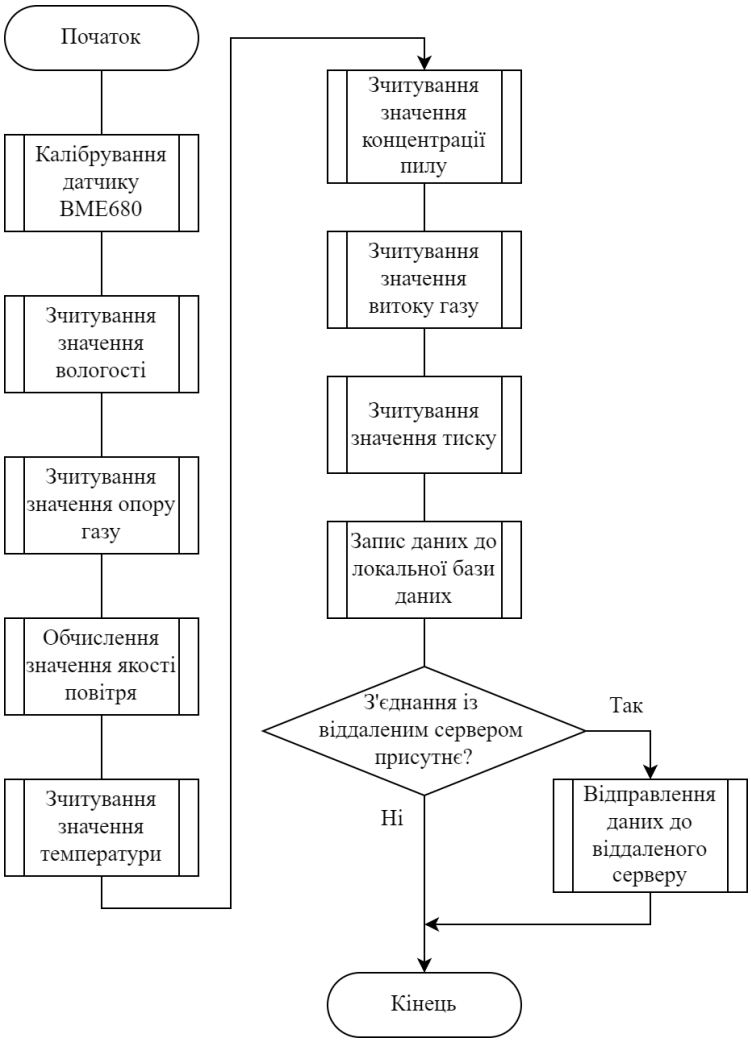


Рисунок 2.8 — Блок-схема загального алгоритму роботи програмного забезпечення мікроконтролера

Збереження даних до локальної бази даних може відбуватися через певні проміжки часу, або через певну кількість вимірів для зменшення об’єму бази даних та зменшення навантаження на мікроконтролер [34].

Надсилання даних до віддаленого серверу потребує з’єднання з інтернетом, що може бути не доступним, тому потрібно реалізувати функціонал, що забезпечить продовження роботи модуля та збереження даних навіть за відсутності з’єднання із сервером.

Також для підтримки з’єднання із сервером у разі його зникнення, програмне забезпечення має автоматично під’єднуватися до серверу.

2.4 Проектування роботи програмного забезпечення Android застосунку

Оскільки Android застосунок знаходиться на рівні презентації даних користувачу, до функціональності програмного забезпечення ставляться відповідні вимоги [35].

Розроблювальний застосунок повинен отримувати дані із віддаленого серверу та відображати їх у зручному та зрозумілому вигляді для користувача. Отримані дані також потребують форматування та графічного виведення для розуміння зчитаних значень упродовж певного періоду.

Після запуску застосунок повинен отримати дані за останній виміри із віддаленого серверу та відобразити це користувачу. Також, за потреби користувач має мати можливість переглянути зчитані дані за певний період у графічному форматі.

Якщо з’єднання із сервером недоступне, або даних за певний період немає, програма також має повідомити про це користувача.

На рисунку 2.9 зображено загальний алгоритм роботи Android застосунку.

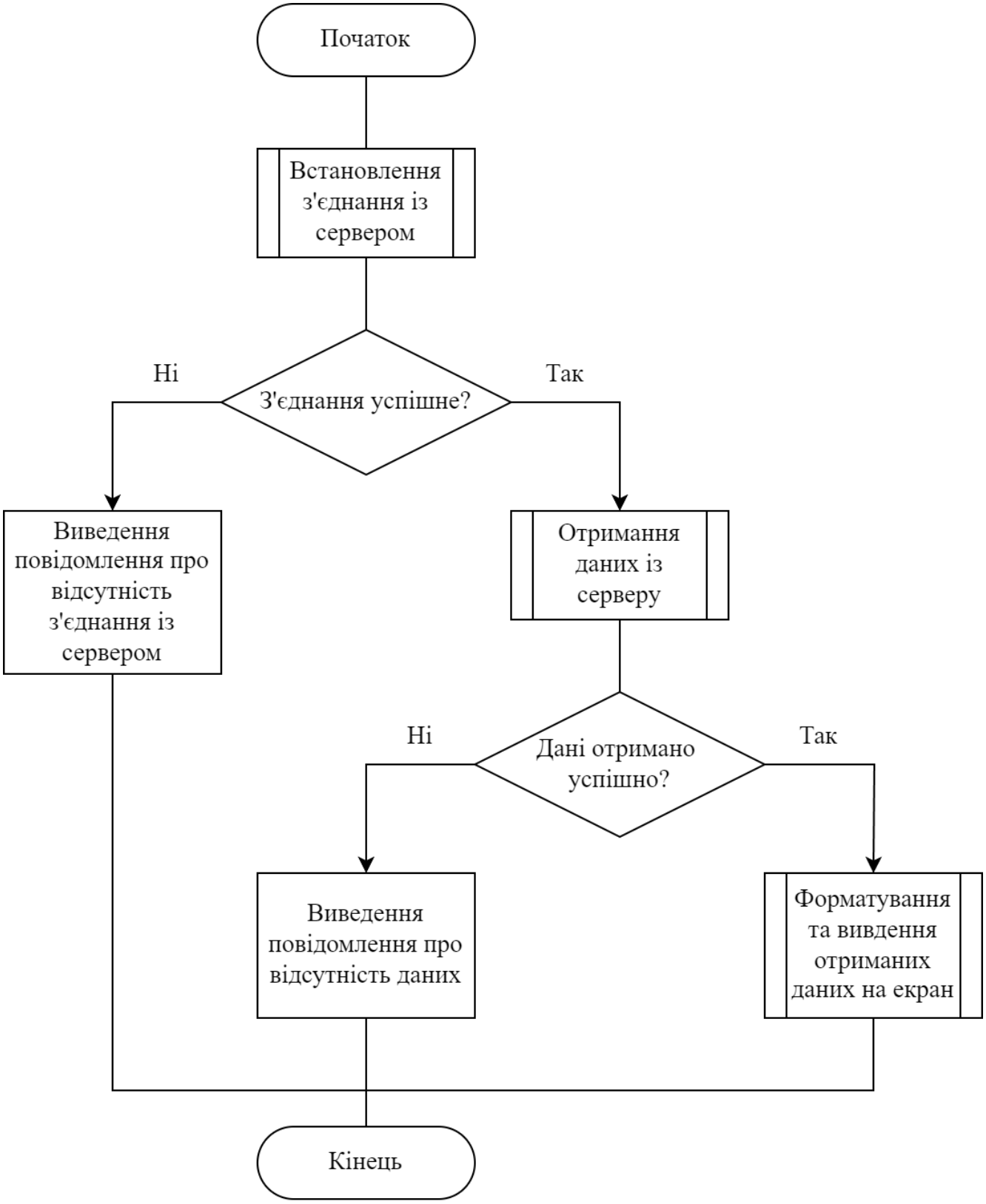


Рисунок 2.9 — Блок-схема загального алгоритму роботи Android застосунку

Також програмне забезпечення Android застосунку повинне відповідати вимогам надійності та повинне мати зручний та зрозумілий користувацький

інтерфейс [36]. У разі виникнення помилки на сервері, програма повинна відобразити відповідне повідомлення користувачу та продовжити свою роботу.

Графічне виведення отриманих даних може включати в себе як показ графіків за певний період вимірів, так і числове відображення зчитаних показників температури, вологості, концентрації пилу та показника якості повітря.

2.5 Висновки

У цьому розділі було обґрунтовано вибір мов програмування та середовищ інтегрованої розробки програмного забезпечення. Також було проведено аналіз існуючих рішень для вибору основних бібліотек для роботи з датчиками пристрою та іншими функціональними вимогами, як от реалізація локальної бази даних.

Було описано основні функціональні та структурні вимоги розроблювального програмного забезпечення та визначено основні алгоритми роботи мікроконтролерів та інших пристроїв для подальшої реалізації програмного забезпечення.

Оскільки система моніторингу компонентів довкілля є системою інтернету речей, було розподілено функціональні обов’язки кожного з компонентів системи відповідно до визначених рівнів структури інтернету речей.

Отже, проведений аналіз та розробка алгоритмів програмного забезпечення стане фундаментом для розробки самого програмного забезпечення та його імплементації у пристрої системи моніторингу компонентів довкілля.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

3.1 Принцип роботи програмного забезпечення модуля зчитування компонентів навколишнього середовища

Програмне забезпечення модуля зчитування компонентів навколишнього середовища написане мовою Python. Програма виконує зчитування даних з усіх датчиків, а саме з датчику пилу Dust Sensor, датчику витоку газу MQ-135 та датчика газу, температури та вологості BME 680.

Датчик BME 680 має свою бібліотеку для зчитування даних з нього та перетворення зчитаних даних в інформацію, яку може читати та розуміти людина.

Програма приймає два аргументи під час запуску: тривалість калібрування сенсорів у секундах та період відправлення даних у хвилинах. Якщо аргументи не було надано, то використовується значення за замовчуванням, а саме 300 секунд калібрування та 15 хвилин період.

Першим етапом є з’єднання з локальною базою даних. Цей крок є важливим, адже збереження отриманих значень з датчиків потрібно зберігати навіть у випадку втрати з’єднання із сервером.

Другим етапом є з’єднання із мікроконтролером Raspberry Pico, де встановлено додатково два сенсори: сенсор пилу та сенсор витоку газу. Якщо з’єднання пройшло успішно, то програма продовжує роботу, якщо ні, то виводиться відповідне повідомлення та завершується виконання програми. Така поведінка є виправданою тим, що зчитані дані з мікроконтролера є важливими та потребують постійного оновлення.

Третім етапом є з’єднання із віддаленим сервером через протокол MQTT. Якщо з’єднання пройшло успішно, то програма повідомляє про це та продовжує роботу. Якщо з’єднання не вдалось, програма також продовжує роботу, але при цьому намагається відновити з’єднання в окремому потоці. Після виклику функції

для з’єднання із сервером, відкривається окремий потік, що керує повторним з’єднанням у разі його втрати.

Далі відбувається ініціалізація сенсора та його налаштування. Спочатку створюється об’єкт класу BME680, при створенні якого задається основна адреса для зчитування даних мікроконтролера Raspberry Pi, якщо виникає помилка, то використовується другорядна адреса.

Для кожного виду даних встановлюється рівень дискретизації, що допомагає збільшити точність вимірів, фільтруючи небажаний шум у сигналі. Доступні наступні рівні дискретизації: 1x, 2x, 4x, 8x, та 16x. Кожен з цих рівнів збільшує точність вимірів, але при цьому збільшує навантаження на модуль. Для сенсора тиску встановлюється розмір фільтру, що дозволяє відкинути короткочасні зміни в зчитуваних даних, що можуть бути зумовлені навколишніми подразниками. Для значення фільтру доступні наступні коефіцієнти: 0, 1, 3, 7, 15, 31, 63, 127 [37].

Також потрібно встановити статус сенсору газу, щоб включити зчитування значення із сенсору газу. Для зчитування опору газу, сенсор обладнаний нагрівачем, в якого також налаштовується температура нагріву та період нагрівання.

Далі для калібрування сенсорів розпочинається зчитування даних з них протягом певного часу, що створює початкові дані для задання базису та подальшого розрахунку коефіцієнта якості повітря.

Після успішного з’єднання з мікроконтролером Raspberry Pico, послідовно зчитуються дані із пристрою BME 680, а саме температура, опір газу, вологість повітря та розраховується загальна якість повітря на основі зчитаних даних [38].

Розрахунок якості повітря відбувається відповідно до базового значення опору повітря, що було отримано з процесу калібрування та вологості повітря, що було визначено заздалегідь [39]. На початку зчитується поточне значення опору газу із сенсорів. Далі розраховується зміщення поточного значення опору до його базису. Те саме робиться для вологості повітря. Якщо зміщення вологості додатне, то розраховується значення оцінки вологості за формулою 3.1.

$$hum_{score} = \left(\frac{100 - hum_{baseline} - hum_{offset}}{100 - hum_{baseline}}\right) * \left(hum_{weighting} * 100\right), \quad (3.1)$$

де hum_{score} – розрахункове значення оцінки вологості;

$hum_{baseline}$ – базове значення вологості;

hum_{offset} – зміщення зчитаного значення вологості від базису;

$hum_{weighting}$ — вагове значення для вологості.

Якщо зчитане значення вологості від'ємне, то розрахунок оцінки значення вологості відбувається за формулою 3.2.

$$hum_{score} = \left(\frac{\left(hum_{baseline} + hum_{offset}\right)}{hum_{baseline}}\right) * \left(hum_{weighting} * 100\right), \quad (3.2)$$

Також розраховується оцінка значення опору повітря. Якщо зчитане значення опору додатне, то оцінка розраховується за формулою 3.3.

$$gas_{score} = \left(\frac{gas_{resistance}}{gas_{baseline}}\right) * \left(100 - \left(hum_{weighting} * 100\right)\right), \quad (3.3)$$

де gas_{score} – розрахункове значення оцінки опору повітря;

$gas_{resistance}$ – зчитане значення опору повітря;

$gas_{baseline}$ – базис опору повітря;

$hum_{weighting}$ – вагове значення для вологості.

Якщо зчитане значення опору повітря від'ємне, то розрахунок оцінки відбувається за формулою 3.4.

$$gas_{score} = 100 - \left(hum_{weighting} * 100\right), \quad (3.4)$$

Значення якості повітря є сумою оцінок вологості повітря та оцінки значення опору повітря.

Остаточне значення якості повітря розраховується за формулою 3.5.

$$air_{quality} = hum_{score} + gas_{score}, \quad (3.5)$$

де $air_{quality}$ – розрахункове значення якості повітря.

Зчитування значення температури також супроводжується поправкою на температуру центрального процесора Raspberry Pi. Спочатку зчитується поточна температура центрального процесору в градусах Цельсія. Далі зчитана температура зберігається у масив, що може містити до 10-и значень. Потім з цього масиву розраховується середнє арифметичне температури. Даний крок потрібен для запобігання некоректного виводу даних через стрибки значення температури процесора. Після цього зчитується значення температури з датчика. Вихідна температура розраховується за формулою 3.6.

$$temp = temp_{raw} - \left(\frac{temp_{cpu} - temp_{raw}}{temp_{factor}}\right), \quad (3.6)$$

де temp – розрахункове значення температури з поправкою на температуру центрального процесору;

$temp_{cpu}$ — середнє арифметичне зчитаних температур процесору;

$temp_{raw}$ — зчитане значення температури з датчику;

$temp_{factor}$ — множник поправки зчитаної температури.

Усі дії, що відбуваються під час роботи програмного забезпечення Raspberry Pi виводяться в окремий файл із штампом дати та часу. Це допомагає слідкувати за роботою модуля та знаходити можливі причини некоректної роботи.

На мікроконтролер Raspberry Pico надсилається керуючий сигнал, після чого очікується відповідь з даними про кількість частинок пилу у повітрі та наявність витоку газу.

Далі дані виводяться на цифровий дисплей та відправляються на віддалений сервер за допомогою протоколу MQTT. Водночас зчитані значення зберігаються у локальну базу даних, що запобігає втраті даних за відсутності з'єднання із сервером.

Виведення інформації на цифровий LCD-дисплей виконується в окремому потоці задля запобігання переривання виконання програми під час виведення даних на дисплей.

У разі втрати з’єднання із віддаленим сервером, окремий потік, що слідкує за наявністю з’єднання із сервером, поновлює його через певні інтервали часу. Така поведінка є зручною, адже не потребує перезапуску модуля для повторного з’єднання із сервером, а також запобігає його примусовому припиненню роботи.

Після закриття програми, з’єднання із сервером та базою даних автоматично закривається.

На рисунках 3.1 та 3.2 зображено алгоритм роботи програми одноплатного комп’ютера Raspberry Pi.

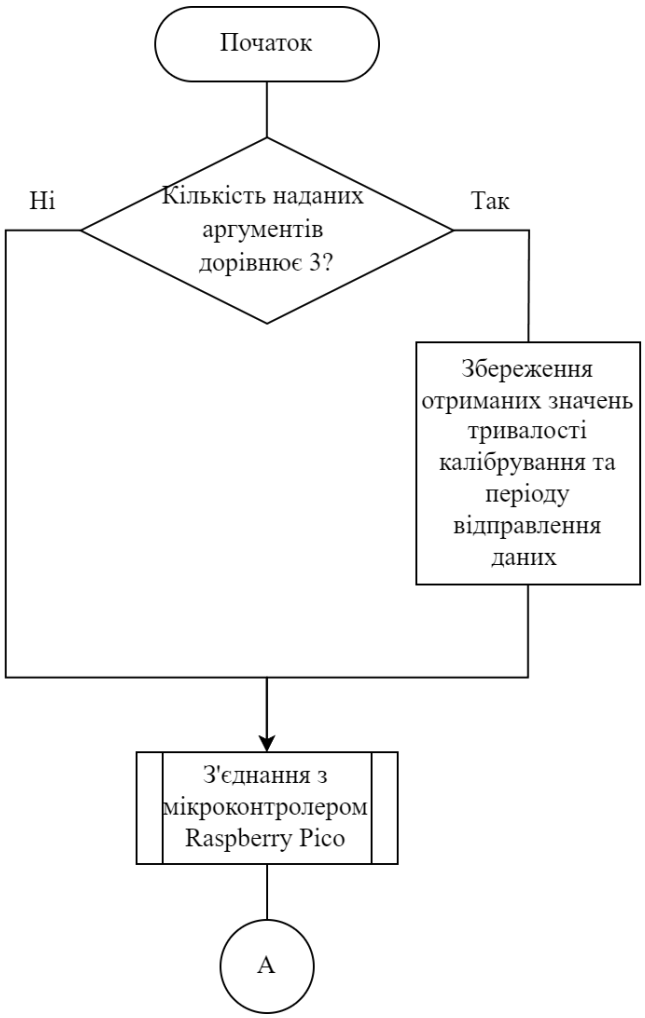


Рисунок 3.1 — Блок-схема алгоритму роботи програми одноплатного комп’ютера Raspberry Pi

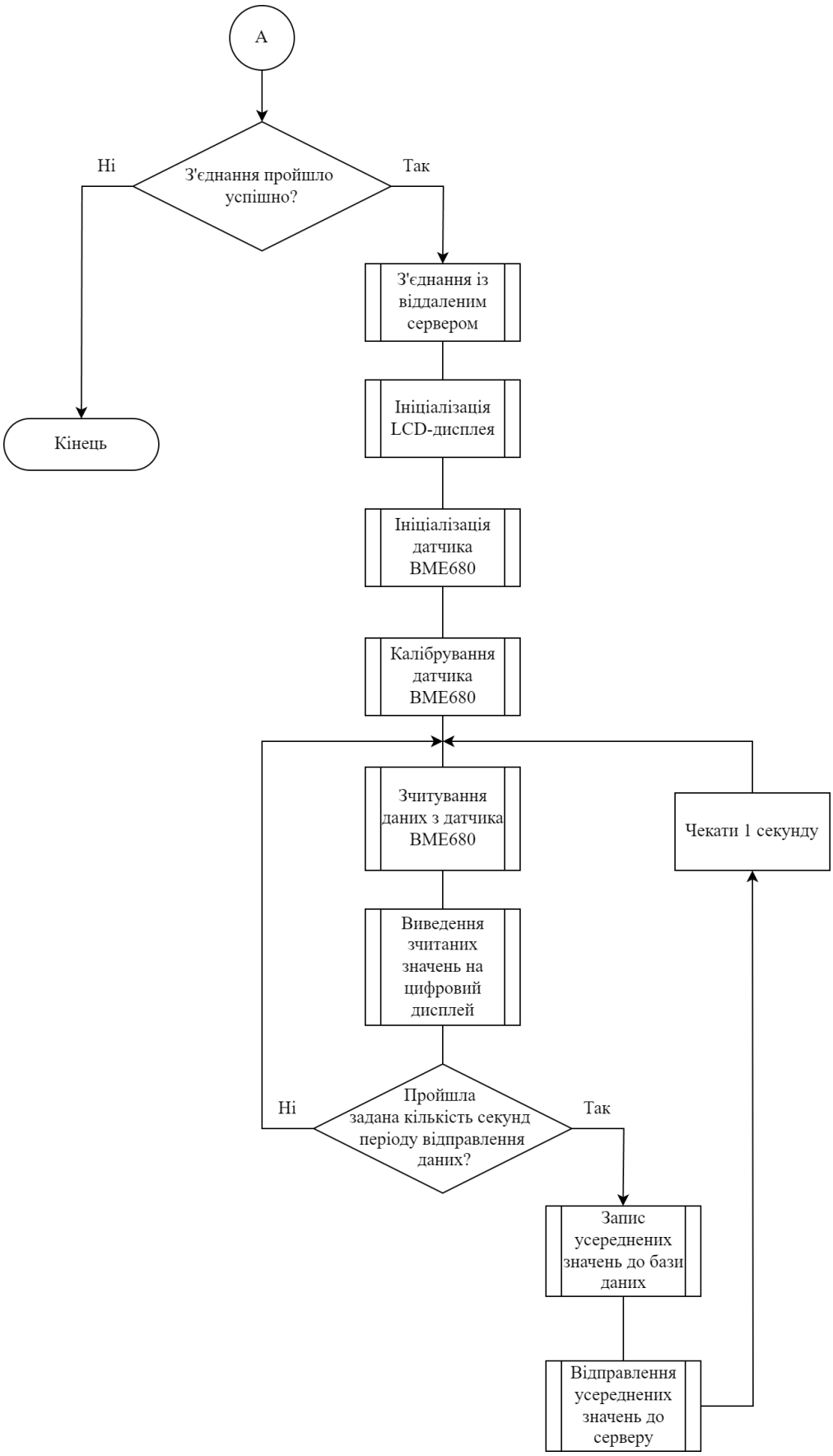


Рисунок 3.2 — Блок-схема алгоритму роботи програми одноплатного комп’ютера Raspberry Pi

Оскільки модуль витоку газу MQ-135 та датчик пилу GP2Y1010AU0F оперують аналоговими значеннями, було додатково використано мікроконтролер Raspberry Pico, що має аналогові входи [40]. Відповідно до цього було запрограмовано мікроконтролер таким чином, аби відбувалося коректне зчитування даних та їх конвертація в зрозумілі для людини величини.

Розглянемо отримувані показники з датчика пилу GP2Y1010AU0F. Спочатку відправляється керуючий сигнал логічної одиниці на порт 22 для ініціалізації зчитування даних з модуля. Далі зчитується значення з аналогового порту 21 у діапазоні від 0 до 65536. Дана поведінка диктується функцією дискретизації аналогового значення read_u16() класу ADC бібліотеки machine. Після успішного зчитування значення з датчика пилу, відправляється керуючий сигнал логічного нуля на порт 22 для припинення зчитування.

Після отримання дискретизованого значення відбувається його фільтрація для запобігання раптових змін у зчитуваному значенні. Для фільтрації та корегування значення концентрації пилу використовується буфер для 10-и значень, який початково заповнюється значення з першого читання. Далі розраховується сума всіх значень буфера. При наступних зчитувань перша комірка буфера віднімається від суми та витісняється з буфера. На місце останнього елемента стає зчитане значення. Таким чином розраховується середнє арифметичне з 10-и попередніх зчитаних значень для запобігання різких змін, що можуть бути викликані неточностями вимірів, або зовнішніми подразниками.

Після корегування зчитаного значення потрібно його перетворити в зрозуміле для людини. Для цього спочатку треба отримати значення напруги в мілівольтах за наступною формулою 3.7.

$$voltage = \left(\frac{voltage_{sys}}{65536}\right) * ad_{value} * 11, \quad (3.7)$$

де voltage – розрахункове значення напруги в мілівольтах;

$voltage_{sys}$ – базове значення напруги, від якої живиться мікроконтролер у мілівольтах;

ad_{value} – зчитуване дискретизоване значення.

Далі робиться поправка на напругу при концентрації пилу 0 мкг/м3, що дорівнює 400 мілівольтам [41]. Для отримання остаточного результату отримана напруга множиться на коефіцієнт перетворення, що дорівнює 0,2 (мкг/м3)/мВ [42].

Концентрації пилу розраховується за формулою 3.8.

$$density = (voltage - 400) * 0{,}2, \quad (3.8)$$

де density – концентрація пилу в мікрограмах на метр кубічний.

Також з датчика витоку газу MQ-135 зчитується булеве значення наявності витоку газу.

Після конвертації зчитаного значення очікується керуючий сигнал по послідовному порту USB. Якщо було отримано сигнал 1, тоді виконується відправлення даних.

Такий підхід дозволяє точно керувати мікроконтролером та не забивати послідовний порт, через який надсилаються зчитані дані.

Під час роботи мікроконтролер повинен постійно перевіряти надіслані повідомлення з послідовного порту на наявність керуючого сигналу.

Також, даний підхід дозволяє не перевантажувати мікроконтролер та датчики, адже коли керуючий сигнал не надсилається, дані з датчиків концентрації пилу та наявності витоку газу не зчитуються.

Блок-схема алгоритму роботи програмного забезпечення мікроконтролера Raspberry Pico зображено на рисунку 3.3.

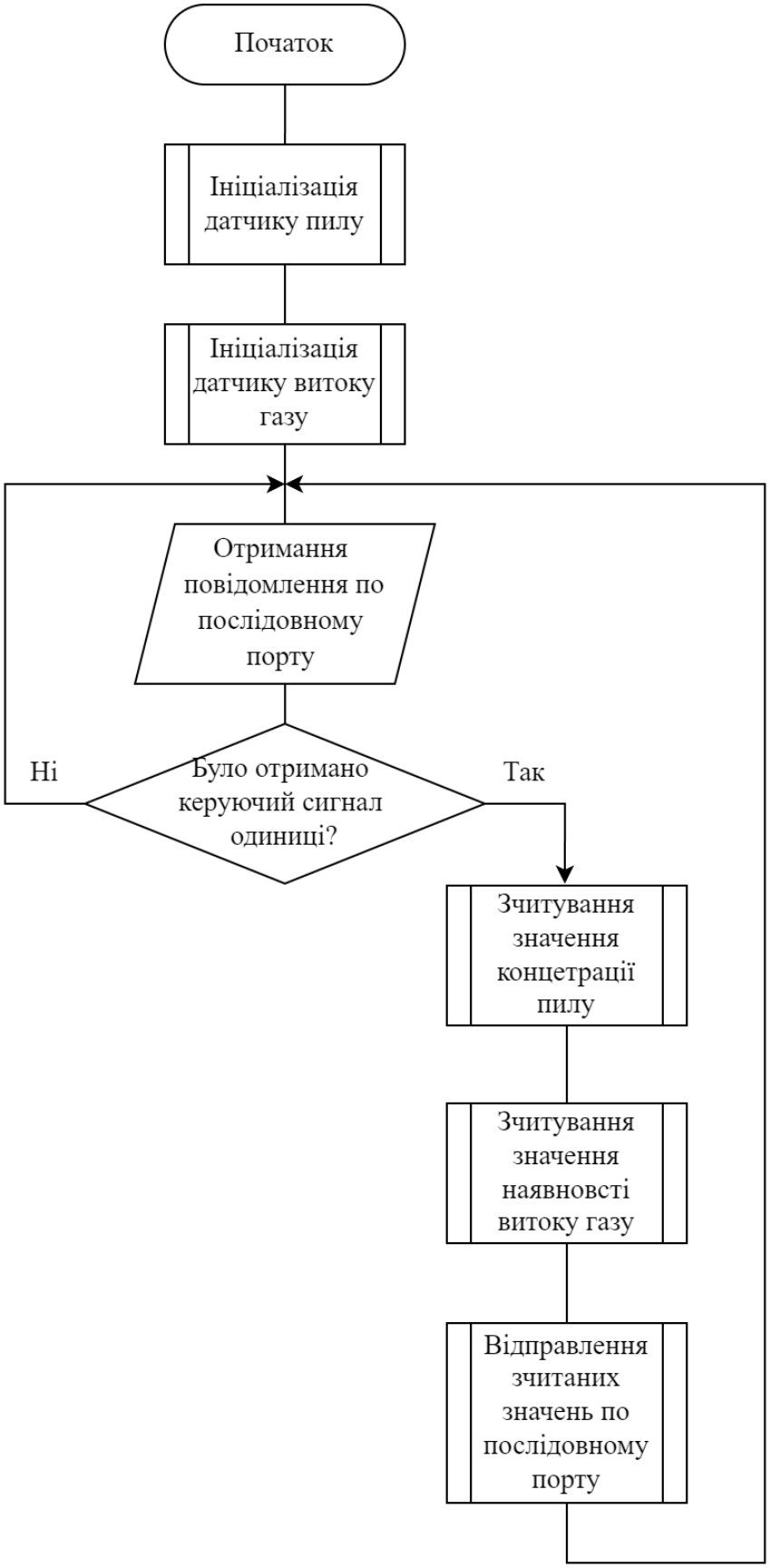


Рисунок 3.3 — Блок-схема алгоритму роботи програмного забезпечення мікроконтролера Raspberry Pico

3.2 Принцип роботи програмного забезпечення Android застосунку

Android застосунок структурно складається з трьох модулів [43]:

– модуль app — цей модуль відповідає за інтерфейс користувача, взаємодію з ним та вивід інформації на екран;

– модуль domain — відповідає за бізнес-логіку додатку, тобто те, як отримані дані опрацьовуються;

– модуль data — відповідає за отримання даних, наприклад, з віддаленого серверу.

Кожен модуль відповідає за свій функціонал та пов’язаний з іншими лише через абстракції, наприклад, інтерфейси [44]. Така модель називається чистою архітектурою. На рисунку 3.1 зображено діаграму дизайну проектування чистої архітектури [45].

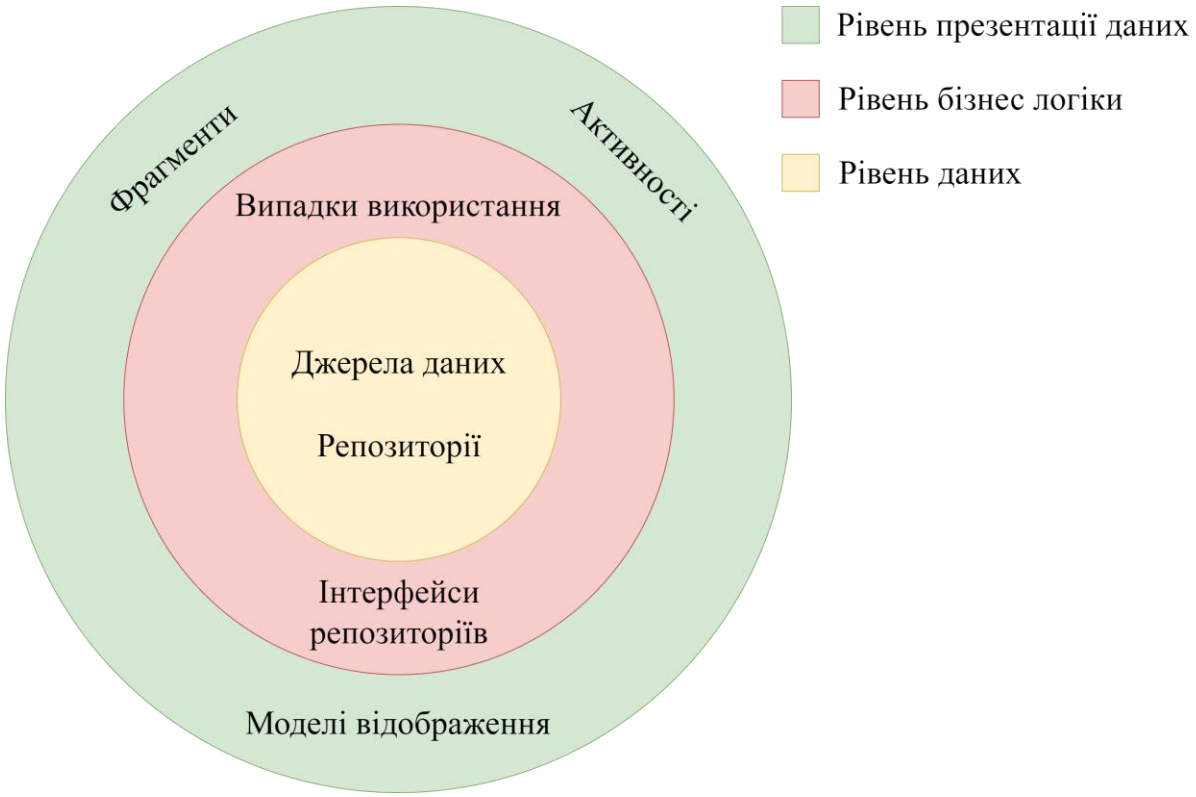


Рисунок 3.4 — Модель чистої архітектури

У дані моделі кожен компонент є незалежним та відповідає лише за певну функціональність. Також кожен із компонентів взаємодіє з тими, що знаходяться нижче в ієрархії через класи та інтерфейси [46]. До прикладу, компонент app взаємодії з компонентом domain через випадки використання, а компонент domain у свою чергу взаємодіє з компонентом data через інтерфейси репозиторіїв.

Модуль data під'єднується до віддаленого сервера та надсилає до нього HTTP GET-запити за потреби через Rest API. Функціонал отримання даних із сервера реалізовано за допомогою безкоштовної бібліотеки з відкритим кодом Retrofit2. Дана бібліотека реалізовує автоматичне з'єднання із сервером за протоколом HTTP, чи HTTPS та опрацьовує запити та відповіді під час обміну даними.

Модуль data також структурно поділений на кілька частин. Першою частиною є підмодуль sources, або джерела [47]. В даному підмодулі реалізовується логіка доступу до джерела даних, як от база даних, чи віддалений сервер. У даному випадку було створено клас AirQualityDataSourceImpl, який реалізовує інтерфейс AirQualityDataSource для доступу до даних на сервері.

Другою частиною модуля data є репозиторії, що використовуються для обробки отриманих даних з підмодуля source та частково перевіряють вхідні дані на правильність [47]. Було створено клас AirQualityRepositoryImpl, який реалізовую інтерфейс AirQualityRepository та наслідує абстрактний клас BaseRepository, в якому описана логіка ловлі та обробки помилок, що можуть виникати під час некоректно переданих даних.

Модуль domain містить у собі так звані use cases, що є випадками використання певного функціоналу наданого модулем data з подальшою обробкою результату [48]. Об'єкт цього класу при створенні отримує в якості аргументу інтерфейс потрібного репозиторію. Даний підхід залежності реалізовує інверсію залежності, що є однією з парадигм програмування SOLID. Було створено три класи: GetAirQualityByDateUseCase, GetCurrentAirQualityUseCase та GetPeriodAirQualityUseCase, кожен з яких реалізовує конкретну логіку часткового використання функціоналу, що надається через інтерфейс AirQualityRepository.

Також кожен із цих класів наслідує базовий клас BaseUseCase, що надає логіку обробки помилок та винятків програми, а також дозволяє виконувати функціонал бізнес-логіки асинхронно за допомогою так званих Coroutine.

Модуль app побудований згідно парадигми MVVM, що розділяє обов'язки функціоналу інтерфейсу користувача між View – те, що користувач безпосередньо бачить — , ViewModel – логіка відображення даних для View – та Model – модель даних, що відображається у View за допомогою ViewModel [48].

Для відображення даних було створено два екрани: домашній екран та екран відображення графіків. На домашньому екрані відображаються останні дані, що було отримано із сервера, а саме показник якості повітря у відсотках, температура у градусах Цельсія, вологість у відсотках, концентрація пилу в мікрограм на метр кубічний, стан витоку газу та тиск у гектопаскалях. На екрані відображення графіків показуються дані за певний період часу: за 5 діб, за 10 діб, за 30 діб; та погодинно за вказану дату.

Також варто відзначити систему dependency injection (ін'єкції залежностей), що дозволяє абстрагувати залежності між підмодулями та винести логіку створення об'єктів класів програми. Для реалізації dependency injection було використано бібліотеку Dagger2.

Модуль app поділено на підмодулі feature та common. Підмодуль common містить у собі базові класи, компонент реалізації ін'єкцій залежностей, класи діалогових вікон та інше. Підмодуль feature містить у собі класи реалізації інтерфейсу користувача, що включає вивід отриманих даних з модуля domain та логіка взаємодії з користувачем.

Застосунок складається з двох основних екранів, між якими користувач може перемикатися через нижнє навігаційне меню. Першим екраном є домашній екран, на якому відображаються останні зчитані дані з датчиків. Після завантаження застосунку відкривається початковий екран у якому здійснюється Get-запит протоколу HTTP для отримання останні зчитаних даних з датчиків, що зберігаються на сервері. Якщо дані було отримано успішно, то вони

відображаються на екрані користувачу. У разі відсутності з’єднання з інтернетом, або із віддаленим сервером, застосунок повідомляє про це користувача та пропонує йому спробувати відправити запит знову. Якщо сервер надсилає помилку, про це застосунок також повідомляє користувача.

Другим екраном застосунку є екран графічного відображення даних з датчиків за певний період часу. Даний екран має два режими роботи: відображення гістограм зчитаних значень датчиків за певний період днів (5, 10, 30 днів) та відображення гістограми отриманих значень за певний день погодинно.

Після відкриття цього екрану автоматично відображаються гістограми значень температури, вологості, показника якості повітря та концентрації пилу за 5 попередніх днів у середньому за день. Користувач може обрати період відображення кількості днів з випадаючого списку, що має варіанти 5 днів, 10 днів та 30 днів.

Для відображення графіку погодинно за певний день, користувач повинен обрати режим «View by hours», після чого автоматично відкриється календар для вибору дати, для якої потрібно відобразити дані. Також, якщо користувач захоче змінити дату відображення даних, то він повинен буде натиснути на відповідно кнопку з відображенням дати, після чого відкриється календар для вибору дати.

Екран відображення графіків також надсилає Get-запити з параметрами відповідно до режиму роботи. Якщо дані успішно отримано, вони автоматично відображаються у вигляді гістограм. Якщо сервер не є доступний, або інтернет-з’єднання відсутнє на пристрої, застосунок повідомляє про це користувача та пропонує повторити запит знову. Якщо сервер повертає помилку, то відповідна інформація про це виводиться користувачу. Дані, що запитує застосунок, можуть не існувати на сервері, у такому разі застосунок повідомляє про це користувача.

Також варто зазначити, що для відображення даних у графічному вигляді було використано бібліотеку з відкритим кодом MPAndroidChart, що значно спрощує процес відображення даних у графічному вигляді та надає функціонал редагування графіків.

Кожен з екранів застосунку складається з двох класів: фрагмент та модель відображення. Фрагмент керує всією логікою інтерфейсу користувача, водночас модель відображення відповідає за взаємодію із модулем domain для отримання даних та наданням їх фрагменту.

Фрагмент у свою чергу відстежує зміни, що надсилає модель відображення. Після зміни даних, графічний інтерфейс оновлюється. Причиною такого розділення функціоналу застосунку є короткий період існування фрагменту, порівняно з моделлю відображення. Якщо навіть фрагмент знищується, або заново створюється, наприклад, після повороту екрану, дані не втрачаються, адже вони зберігаються в модулі відображення.

Головний екран складається з фрагменту HomeFragment та моделі відображення HomeViewModel. Після створення фрагменту, він починає відстежувати зміни даних, що надходять з моделі відображення. У свою чергу модель відображення після створення виконує запит до серверу за посередництвом класу GetCurrentAirQualityUseCase, що є частиною модуля domain. Метод даного класу повертає об’єкт класу AirQuality, що складається з властивостей date типу Date, temperature типу Double, humidity типу Double, airQualityScore типу Double, dustConcentration типу Double та gasLeaks типу Boolean. Запит виконується асинхронно, щоб не завантажувати основний потік виконання процесу програми. HomeViewModel має методи, що викликаються відповідно до якоїсь події: дані отримано успішно, було отримано помилку від серверу, інтернет-з’єднання відсутнє та сталася невідома помилка. Відповідно до кожного випадку виконується відповідна дія відображення помилки користувачу, або ж відображення отриманих даних. Також окремо оброблюється процес відправлення запиту та отримання відповіді, під час якої відображається діалогове вікно завантаження, аби не перенавантажувати пристрій та повідомляти користувача про те, що відбувається запит до серверу.

Логіка екрану відображення графіків подібна до логіки домашнього екрану із доданим функціоналом обробки даних для їх коректного відображення у вигляді

гістограм та обробкою кількох різних запитів до серверу та взаємодією із користувачем. Даний екран складається з фрагменту ChartFragment та моделі відображення ChartViewModel.

Після завантаження фрагмент виконує метод запиту даних за останні 5 днів через модуль відображення даних. У свою чергу модель відображення даних виконує запит через клас GetPeriodAirQualityUseCase, що визначений в модулі domain та повертає список об’єктів класу AirQuality. Якщо дані було отримано успішно, фрагмент відображає їх через гістограми для температури, вологості, якості повітря та концентрації пилу окремо. Якщо виникає помилка, то користувачу виводиться повідомлення про це.

Як тільки користувач обере інший період виводу даних, фрагмент виконає відповідний метод моделі відображення для отримання даних за цей період. Також, якщо буде обрано інший режим роботи екрану, що передбачає вивід графіка за певну дату погодинно, фрагмент покаже діалогове вікно вибору дати. Після обрання певної дати користувачем, фрагмент виконає відповідний метод моделі відображення. Якщо дані існують, то сервер надішле їх, а фрагмент відобразить на екрані. В іншому випадку, буде відображено помилку, або повідомлення на графіках про те, що дані відсутні.

На рисунках 3.5-3.7 зображено блок-схему алгоритму роботи Android застосунку.

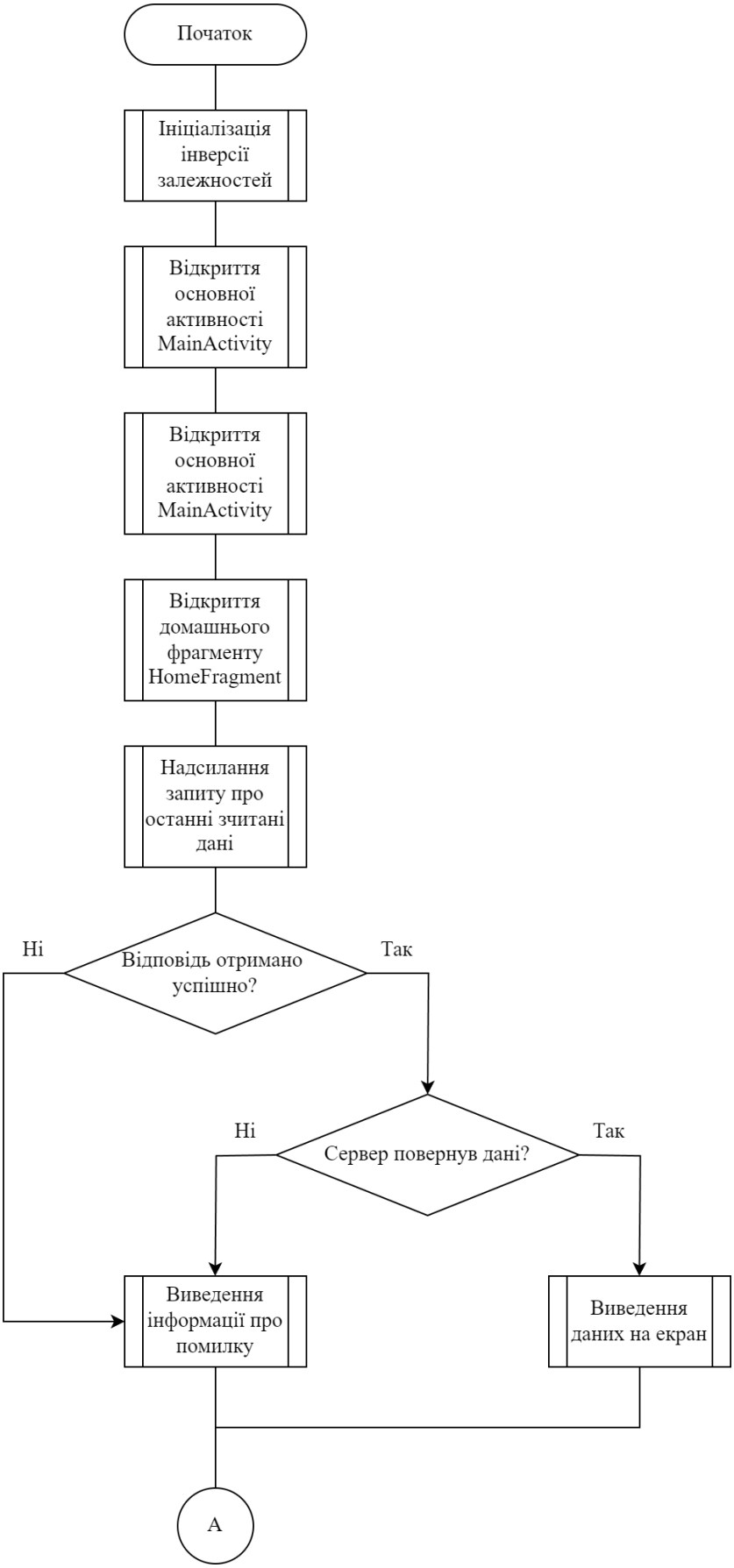


Рисунок 3.5 — Блок-схема алгоритму роботи Android застосунку

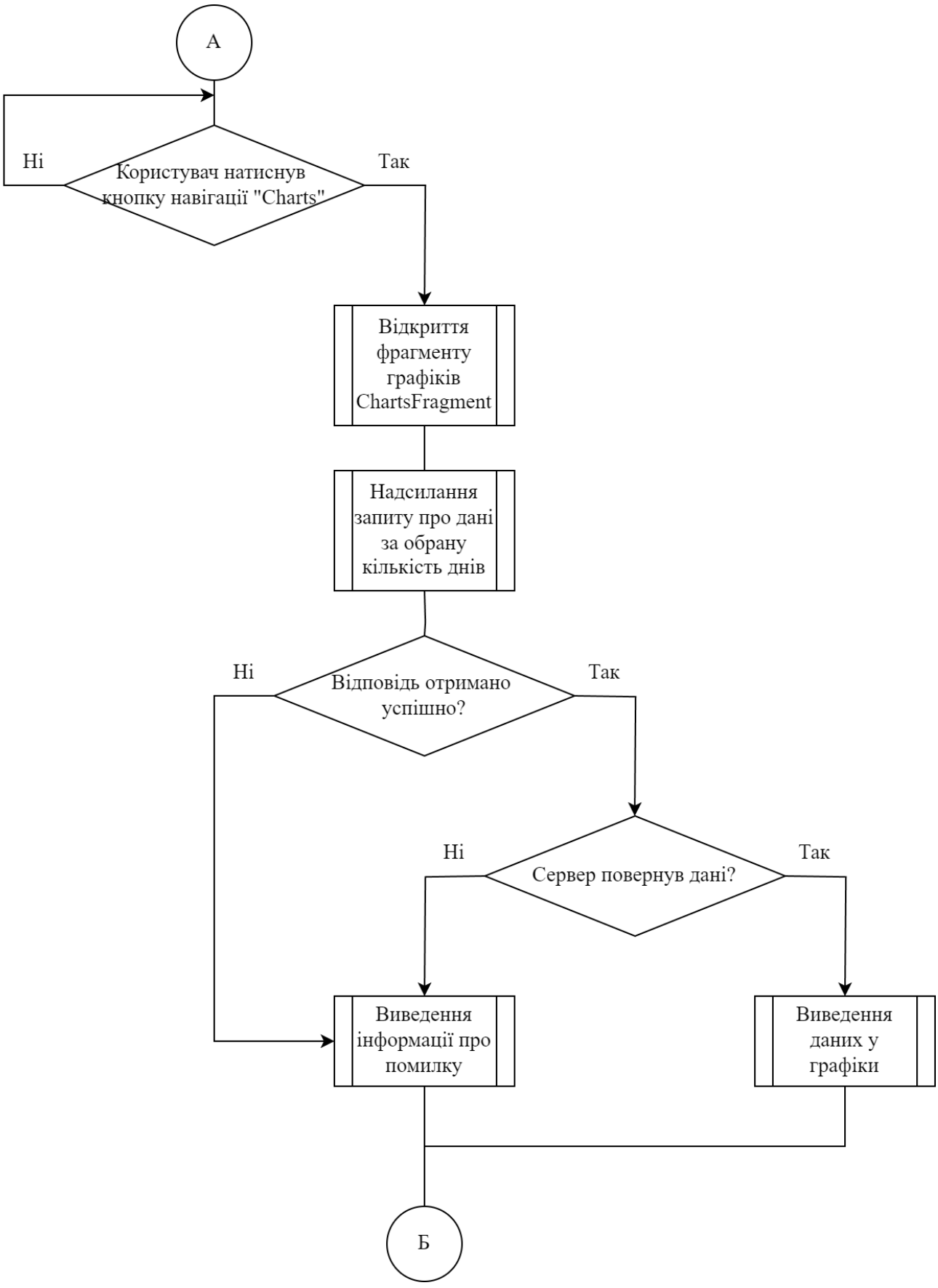


Рисунок 3.6 — Блок-схема алгоритму роботи Android застосунку

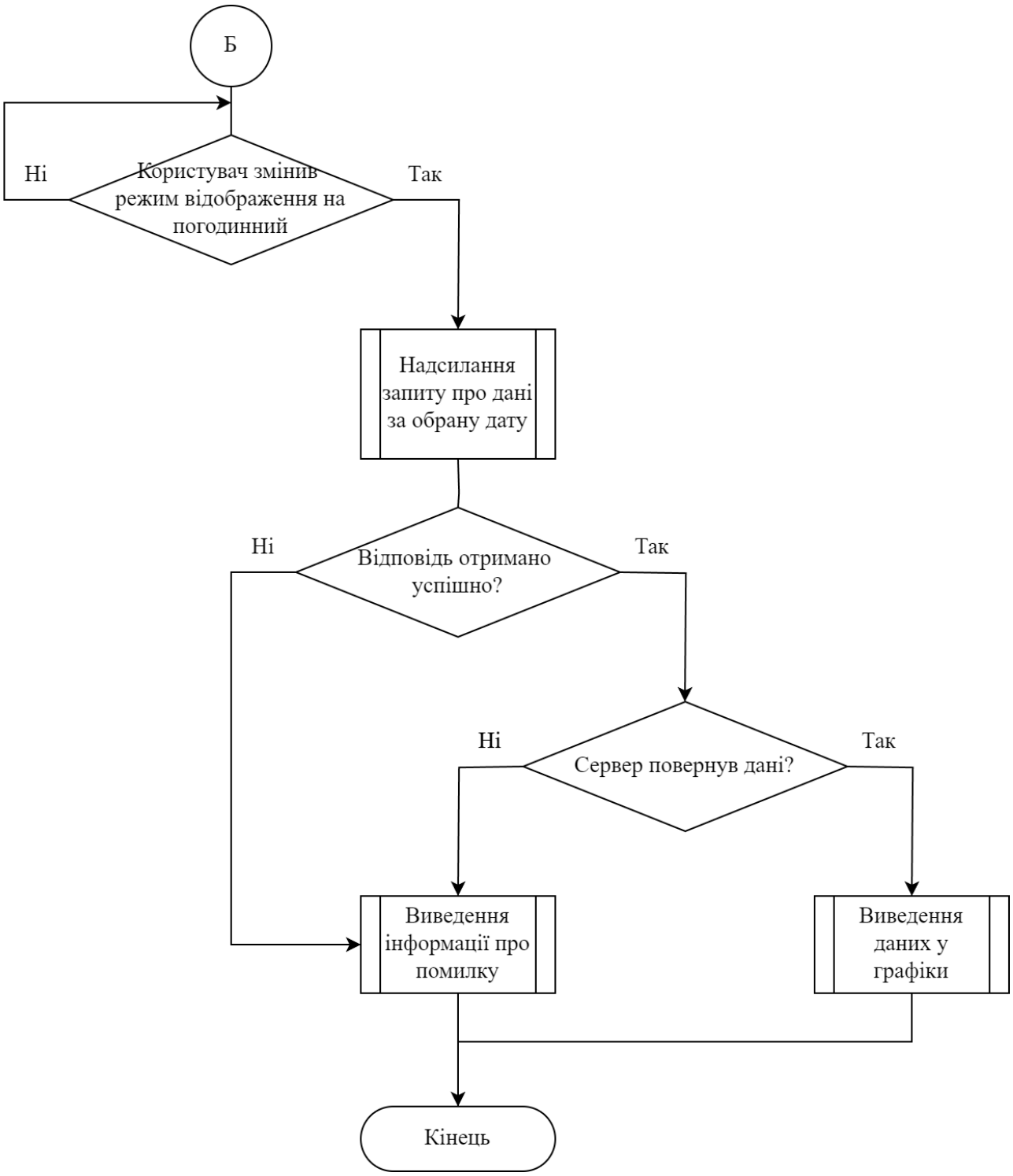


Рисунок 3.7 — Блок-схема алгоритму роботи Android застосунку

3.3 Результати роботи програмного забезпечення мікроконтролера

Після запуску одноплатного комп'ютера Raspberry Pi, програмне забезпечення запускається автоматично. Це досягається за допомогою програми-планувальника Cron, що запускається автоматично зі стартом операційної системи Rasbian. На рисунку 3.8 зображено вміст файлу конфігурації програмного забезпечення Cron.

```
  GNU nano 5.4                                  /tmp/crontab.uQtZGl/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
@reboot sleep 30; sudo python /home/vlad/Diploma/AirQualityControl.py
```

Рисунок 3.8 — Вміст файлу конфігурації Cron для автоматичного запуску програми

Оскільки програмне забезпечення виводить інформацію про дії, що виконуються, у файл логування, там можна знайти усю потрібну інформацію для підтвердження роботи програми. На рисунку 3.9 зображено вміст файлу логування.

AirQualityControl.py | logs.txt

```
[2024-06-11, 13:06:29]: Disconnected from server with result code: Unspecified error
[2024-06-11, 13:06:29]: Reconnecting to server in 1 seconds...
[2024-06-11, 13:06:35]: timed out. Reconnect to server failed. Retrying...
[2024-06-11, 13:06:35]: Reconnecting to server in 2 seconds...
[2024-06-11, 13:06:42]: timed out. Reconnect to server failed. Retrying...
[2024-06-11, 13:06:42]: Reconnecting to server in 4 seconds...
[2024-06-11, 13:06:51]: timed out. Reconnect to server failed. Retrying...
[2024-06-11, 13:06:51]: Reconnecting to server in 8 seconds...
[2024-06-11, 13:07:04]: timed out. Reconnect to server failed. Retrying...
[2024-06-11, 13:07:04]: Reconnecting to server in 16 seconds...
[2024-06-11, 13:07:20]: Reconnected to server successfully!
[2024-06-11, 13:07:21]: Connected to server
[2024-06-11, 13:21:24]: Data sent successfully
[2024-06-11, 13:36:25]: Data sent successfully
[2024-06-11, 13:51:27]: Data sent successfully
[2024-06-11, 14:06:29]: Data sent successfully
[2024-06-11, 14:09:13]: Program started
[2024-06-11, 14:09:14]: Connected to server
[2024-06-11, 14:09:15]: Data burn in started for 300 seconds
[2024-06-11, 14:13:28]: Disconnected from server with result code: Keep alive timeout
[2024-06-11, 14:13:28]: Reconnecting to server in 1 seconds...
[2024-06-11, 14:13:32]: [Errno 113] No route to host. Reconnect to server failed. Retrying...
[2024-06-11, 14:13:32]: Reconnecting to server in 2 seconds...
[2024-06-11, 14:13:35]: [Errno 113] No route to host. Reconnect to server failed. Retrying...
[2024-06-11, 14:13:35]: Reconnecting to server in 4 seconds...
[2024-06-11, 14:13:42]: [Errno 113] No route to host. Reconnect to server failed. Retrying...
[2024-06-11, 14:13:42]: Reconnecting to server in 8 seconds...
[2024-06-11, 14:13:51]: Reconnected to server successfully!
[2024-06-11, 14:13:51]: Connected to server
[2024-06-11, 14:29:19]: Data sent successfully
[2024-06-11, 14:44:21]: Data sent successfully
[2024-06-11, 14:59:23]: Data sent successfully
[2024-06-11, 15:14:24]: Data sent successfully
[2024-06-11, 15:29:26]: Data sent successfully
[2024-06-11, 15:44:28]: Data sent successfully
[2024-06-11, 15:59:30]: Data sent successfully
[2024-06-11, 16:14:32]: Data sent successfully
[2024-06-11, 16:29:34]: Data sent successfully
[2024-06-11, 16:44:36]: Data sent successfully
[2024-06-11, 16:59:38]: Data sent successfully
[2024-06-11, 17:14:39]: Data sent successfully
[2024-06-11, 17:29:41]: Data sent successfully
[2024-06-11, 17:44:43]: Data sent successfully
[2024-06-11, 17:59:45]: Data sent successfully
[2024-06-11, 18:14:47]: Data sent successfully
[2024-06-11, 18:29:49]: Data sent successfully
[2024-06-11, 18:44:51]: Data sent successfully
```

Рисунок 3.9 — Вміст файлу логування програмного забезпечення Raspberry Pi

Якщо запускати програму через консоль, то також можна побачити зчитані дані з датчиків. На рисунку 3.10 зображено результат виконання програмного забезпечення Raspberry Pi у вікні консолі.

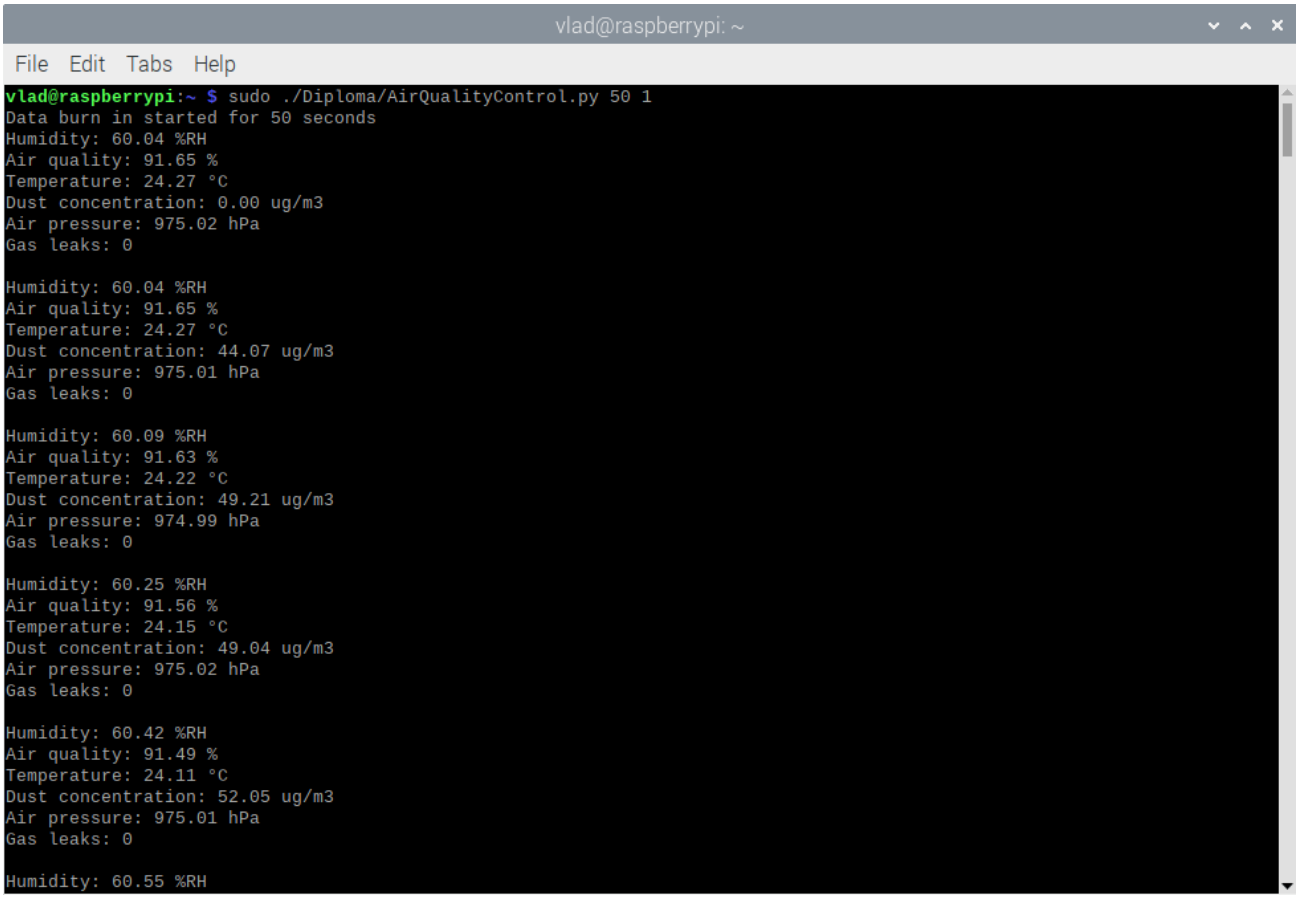
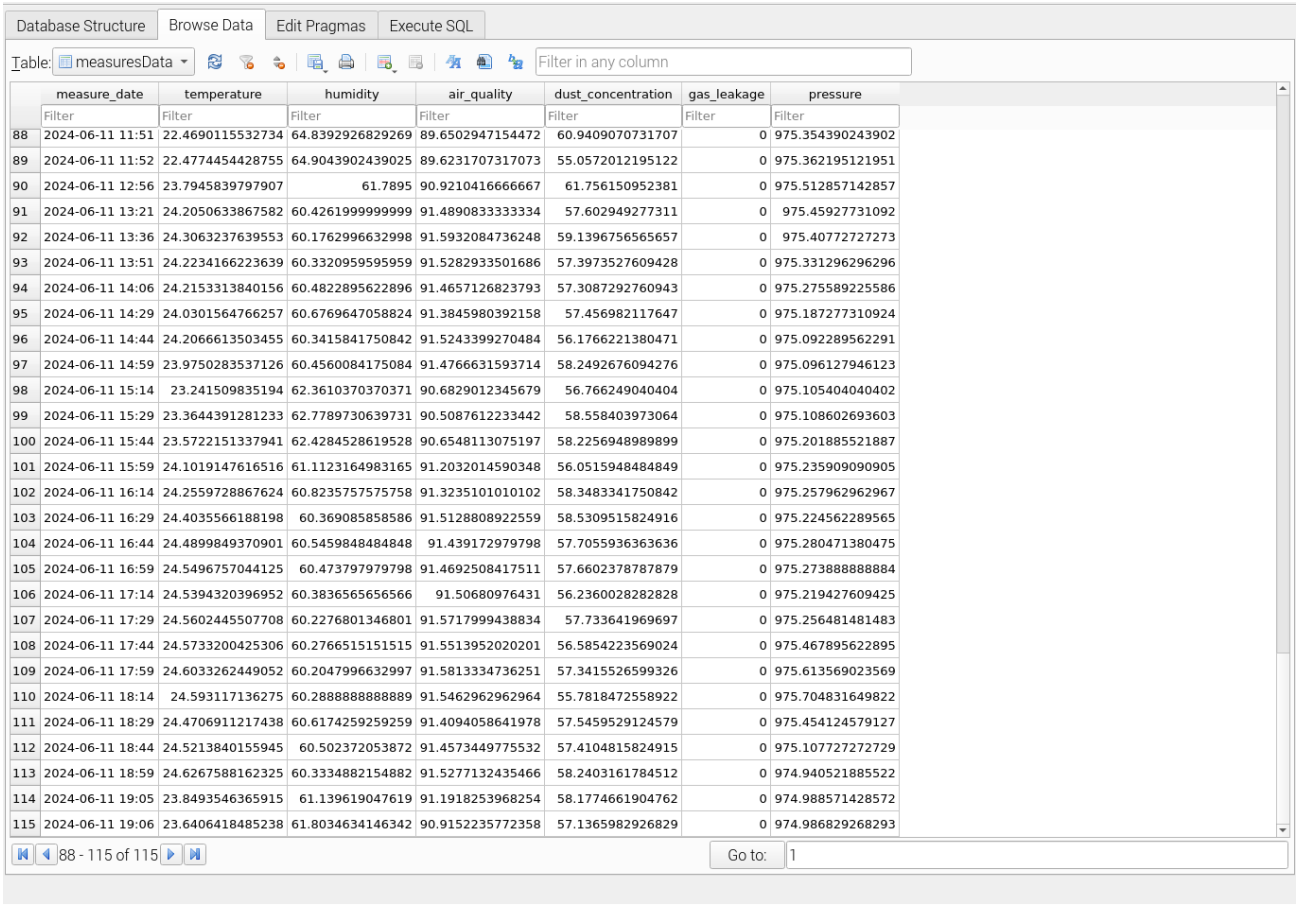


Рисунок 3.10 — Результат виконання програмного забезпечення у консолі

Також через програмне забезпечення DB Browser for SQLite можна переглянути всі наявні записи в локальній базі даних. На рисунку 3.11 зображено вміст локальної бази даних Raspberry Pi.



Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: measuresData — Filter in any column

	measure_date	temperature	humidity	air_quality	dust_concentration	gas_leakage	pressure
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
88	2024-06-11 11:51	22.4690115532734	64.8392926829269	89.6502947154472	60.9409070731707	0	975.354390243902
89	2024-06-11 11:52	22.4774454428755	64.9043902439025	89.6231707317073	55.0572012195122	0	975.362195121951
90	2024-06-11 12:56	23.7945839797907	61.7895	90.9210416666667	61.756150952381	0	975.512857142857
91	2024-06-11 13:21	24.2050633867582	60.4261999999999	91.4890833333334	57.602949277311	0	975.45927731092
92	2024-06-11 13:36	24.3063237639553	60.1762996632998	91.5932084736248	59.1396756565657	0	975.40772727273
93	2024-06-11 13:51	24.2234166223639	60.3320959595959	91.5282933501686	57.3973527609428	0	975.331296296296
94	2024-06-11 14:06	24.2153313840156	60.4822895622896	91.4657126823793	57.3087292760943	0	975.275589225586
95	2024-06-11 14:29	24.0301564766257	60.6769647058824	91.3845980392158	57.456982117647	0	975.187277310924
96	2024-06-11 14:44	24.2066613503455	60.3415841750842	91.5243399270484	56.1766221380471	0	975.092289562291
97	2024-06-11 14:59	23.9750283537126	60.4560084175084	91.4766631593714	58.2492676094276	0	975.096127946123
98	2024-06-11 15:14	23.241509835194	62.3610370370371	90.6829012345679	56.766249040404	0	975.105404040402
99	2024-06-11 15:29	23.3644391281233	62.7789730639731	90.5087612233442	58.558403973064	0	975.108602693603
100	2024-06-11 15:44	23.5722151337941	62.4284528619528	90.6548113075197	58.2256948989899	0	975.201885521887
101	2024-06-11 15:59	24.1019147616516	61.1123164983165	91.2032014590348	56.0515948484849	0	975.235909090905
102	2024-06-11 16:14	24.2559728867624	60.8235757575758	91.3235101010102	58.3483341750842	0	975.257962962967
103	2024-06-11 16:29	24.4035566188198	60.369085858586	91.5128808922559	58.5309515824916	0	975.224562289565
104	2024-06-11 16:44	24.4899849370901	60.5459848484848	91.439172979798	57.7055936363636	0	975.280471380475
105	2024-06-11 16:59	24.5496757044125	60.473797979798	91.4692508417511	57.6602378787879	0	975.273888888884
106	2024-06-11 17:14	24.5394320396952	60.3836565656566	91.50680976431	56.2360028282828	0	975.219427609425
107	2024-06-11 17:29	24.5602445507708	60.2276801346801	91.5717999438834	57.733641969697	0	975.256481481483
108	2024-06-11 17:44	24.5733200425306	60.2766515151515	91.5513952020201	56.5854223569024	0	975.467895622895
109	2024-06-11 17:59	24.6033262449052	60.2047996632997	91.5813334736251	57.3415526599326	0	975.613569023569
110	2024-06-11 18:14	24.593117136275	60.2888888888889	91.5462962962964	55.7818472558922	0	975.704831649822
111	2024-06-11 18:29	24.4706911217438	60.6174259259259	91.4094058641978	57.5459529124579	0	975.454124579127
112	2024-06-11 18:44	24.5213840155945	60.502372053872	91.4573449775532	57.4104815824915	0	975.107727272729
113	2024-06-11 18:59	24.6267588162325	60.3334882154882	91.5277132435466	58.2403161784512	0	974.940521885522
114	2024-06-11 19:05	23.8493546365915	61.139619047619	91.1918253968254	58.1774661904762	0	974.988571428572
115	2024-06-11 19:06	23.6406418485238	61.8034634146342	90.9152235772358	57.1365982926829	0	974.986829268293

88 - 115 of 115 — Go to: 1

Рисунок 3.11 — Вміст локальної бази даних Raspberry Pi

3.4 Результат роботи програмного забезпечення Android застосунку

Запустивши застосунок для операційної системи Android, можна одразу побачити головний екран, де буде відображено останні зчитані дані за умови, що з’єднання із віддаленим сервером присутнє. На рисунку 3.12 зображено головний екран Android застосунку.

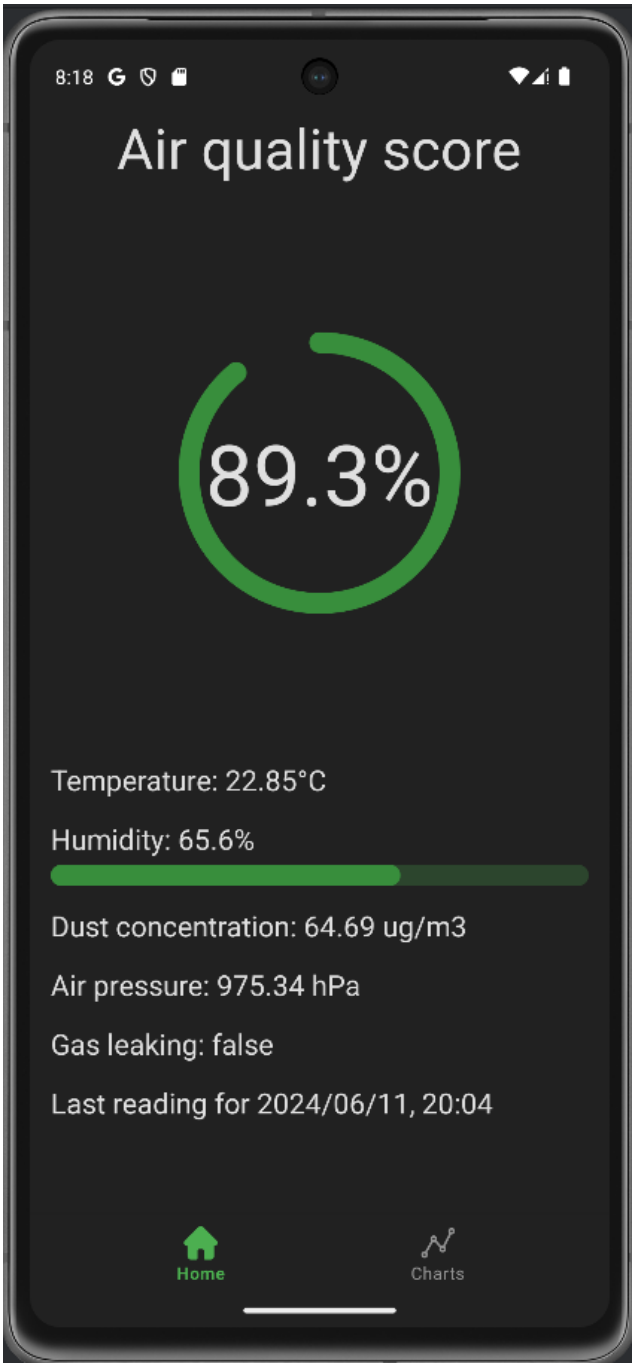


Рисунок 3.12 — Головний екран Android застосунку

Якщо перейти на екран відображення графіків, то користувач побачить 5 графіків з даними за період 5-ти днів. На рисунку 3.13 зображено вигляд екрану відображення графіків.

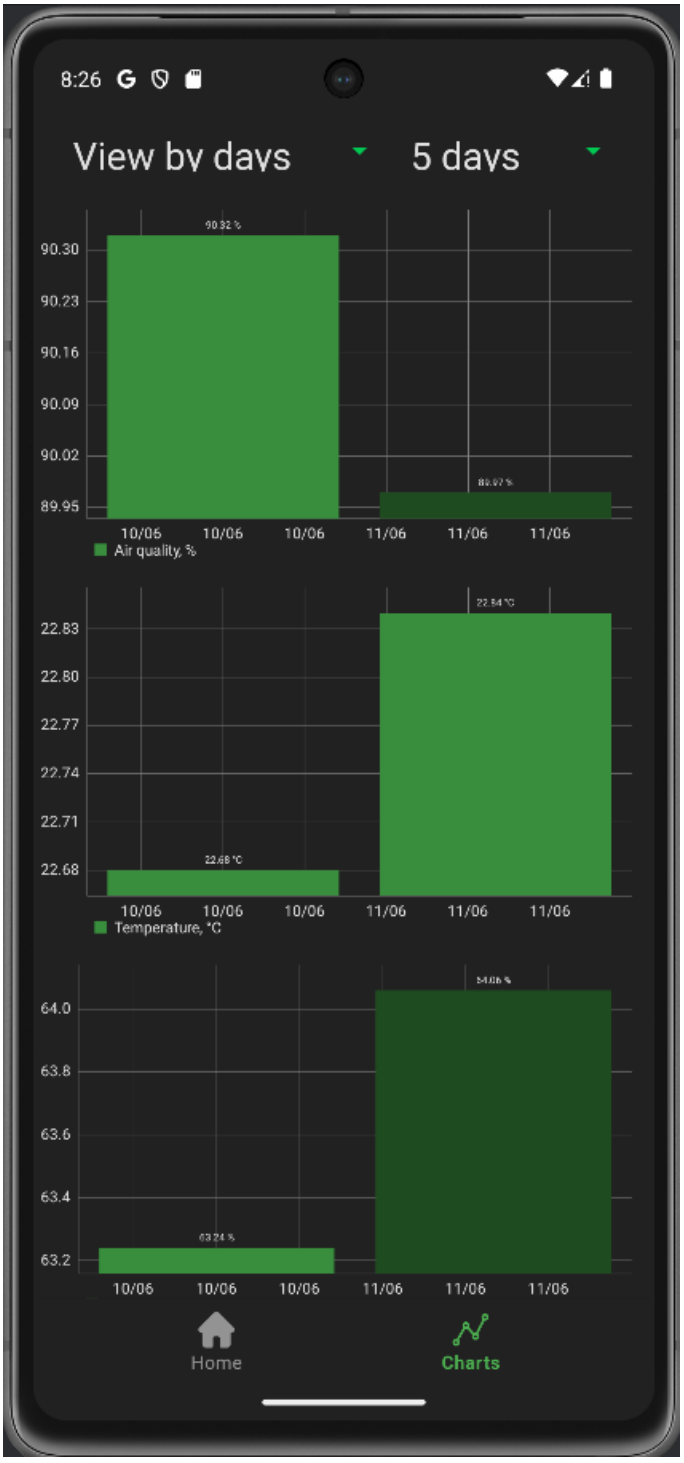


Рисунок 3.13 — Екран відображення графіків зібраних даних

Даний екран має два режими: відображення інформації за період певної кількості днів (5, 10 та 30 днів) та відображення даних за певний день погодинно. Режим погодинного відображення даних зображено на рисунку 3.14.

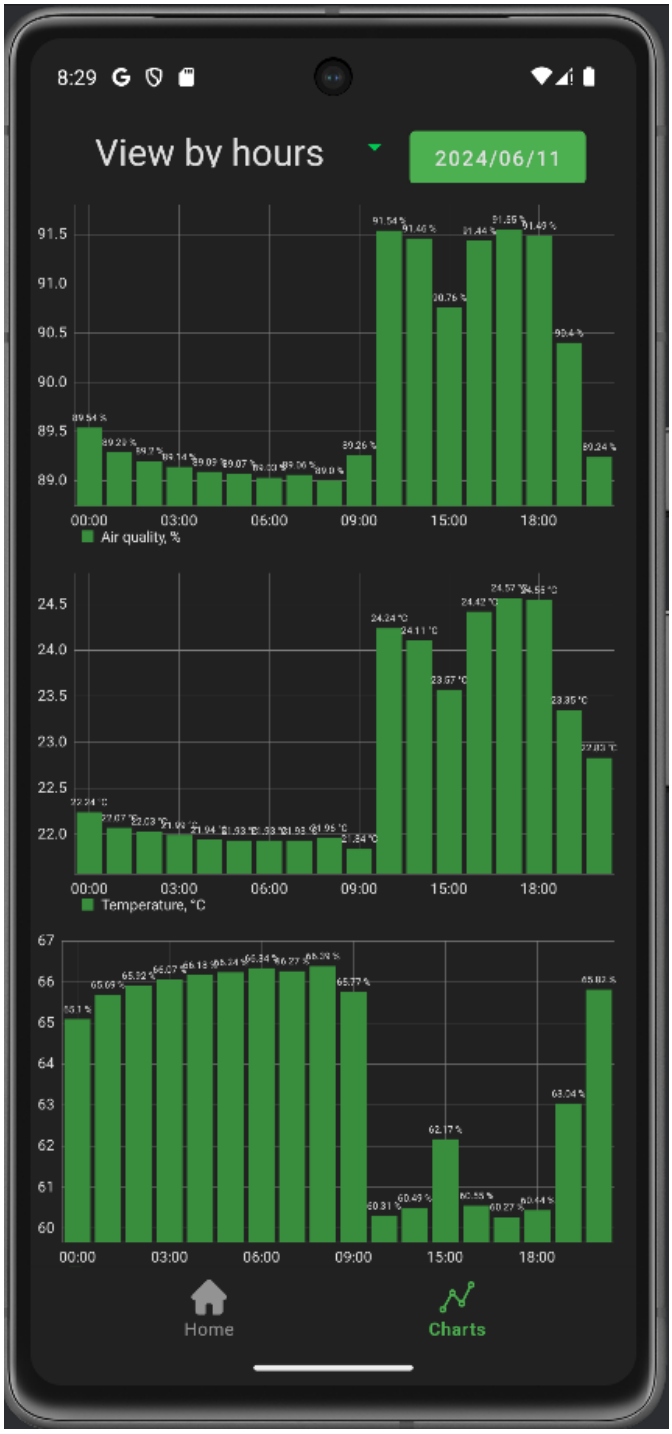


Рисунок 3.14 — Режим відображення графіків погодинно за певну дату

3.6. Висновки

У цьому розділі було детально описано розроблене програмне забезпечення для мікроконтролера Raspberry Pico, одноплатного комп’ютера Raspberry Pi та програмне забезпечення Android застосунку.

Розроблені програми відповідають визначеним функціональним вимогам та розробленим алгоритмам, що були обґрунтовані та описані в розділі 2, а саме збір даних з датчиків, збереження отриманих значень у локальній базі даних та відправлення їх до віддаленого серверу. Описані алгоритми було більш деталізовано.

Android застосунок включає в себе надання доступу до останніх вимірів х датчиків та усереднених значень за певний період або день.

Також розроблене програмне забезпечення було успішно протестовано, а результати його роботи було продемонстровано в цьому розділі.

ВИСНОВКИ

У результаті виконаного дослідження та вивченого матеріалу було розроблено програмне забезпечення мікроконтролерної системи моніторингу компонентів довкілля. Запрограмовані компоненти системи, а саме мікроконтролер Raspberry Pico, одноплатний комп’ютер Raspberry Pi та Android застосунок, виконують усі поставлені функціональні вимоги. Було продемонстровано результати роботи системи моніторингу компонентів довкілля. Також програмне забезпечення було успішно протестовано.

У першому розділі було проведено аналіз предметної області та вивчено існуючі рішення систем моніторингу компонентів довкілля. Також у цьому розділі було запропоновано шлях вирішення досліджуваної проблеми та поетапний план розробки програмного забезпечення.

У другому розділі було запропоновано структуру роботи системи моніторингу компонентів довкілля та визначено загальні алгоритми роботи кожного з компонентів системи.

У третьому розділі було деталізовано алгоритми роботи програмного забезпечення структурних компонентів системи моніторингу компонентів довкілля та розроблено програмне забезпечення відповідно до цього. Також розроблене програмне забезпечення було успішно імплементовано та протестовано.

Отже, у ході дослідження програмної частини мікроконтролерної системи моніторингу компонентів довкілля було успішно розроблено програмне забезпечення, що відповідає вимогам функціональності та надійності існуючих рішень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Pollution and health: a progress update / Fuller R. B. et al. May 17, 2022. DOI: https://doi.org/10.1016/S2542-5196(22)00090-0 (дата звернення 10.03.2024).
2. Air Pollution and Your Health. URL: https://www.niehs.nih.gov/health/topics/agents/air-pollution (дата звернення 10.03.2024).
3. Research on Health Effects from Air Pollution. URL: https://www.epa.gov/air-research/research-health-effects-air-pollution (дата звернення 10.03.2024).
4. Greenhouse gases in air pollution in Major air pollutants. URL: https://www.britannica.com/science/air-pollution/Greenhouse-gases. (дата звернення 11.03.2024).
5. Annual CO_2 emissions. URL: https://ourworldindata.org/grapher/annual-co2-emissions-per-country?time=earliest. 2022 (дата звернення 12.03.2024).
6. Ritchie H., Roser M. CO_2 emissions. How much CO_2 does the world emit? Which countries emit the most? URL: https://ourworldindata.org/co2-emissions (дата звернення 12.03.2024).
7. How Has Air Quality Changed Over Time? Brief History. URL: https://airly.org/en/how-has-air-quality-changed-over-time-brief-history/ (дата звернення 14.03.2024).
8. Thorpe T. Robert Angus Smith. Nature. 1884. pp. 104–105.
9. Brimblecombe P. The Big Smoke: A History of Air Pollution in London Since Medieval Times. Routledge, 1987. pp. 147–160.
10. Bell M. L., Davis D. L., Fletcher T. A Retrospective Assessment of Mortality from the London Smog Episode of 1952: The Role of Influenza and Pollution. Environ Health Perspect. 2004. DOI: 10.1289/ehp.6539 (дата звернення 14.03.2024).
11. Beaver H. Great Britain Committee on Air Pollution: Interim Report. London: HMSO. December 1953. pp. 17–18.
12. Brimblecombe P. The Clean Air Act after 50 years. Weather, 1 November 2006. DOI: 10.1256/wea.127.06 (дата звернення 14.03.2024).

13. Monitoring Methodologies. URL: https://www.airquality.gov.wales/about-air-quality/monitoring/monitoring-methodologies (дата звернення 14.03.2024).

14. Manisalidis I, Stavropoulou E, Stavropoulos A, Bezirtzoglou E. Environmental and Health Impacts of Air Pollution: A Review. Front Public Health. 20 Feb, 2020. DOI: 10.3389/fpubh.2020.00014 (дата звернення 14.03.2024).

15. Air Quality Monitoring System. URL: https://oizom.com/product/polludrone-air-pollution-monitoring/ (дата звернення 17.03.2024).

16. Industrial Air Quality Monitor. URL: https://oizom.com/product/aqbot-industrial-air-quality-monitor/ (дата звернення 17.03.2024).

17. Temtop M10 Air Quality Tester. URL: https://temtopus.com/products/temtop-m10-air-quality-detector-professional-formaldehyde-tvoc-pm2-5-monitor-air-quality-meter. (дата звернення 17.03.2024).

18. Temtop M2000 Air Quality Meter. URL: https://temtopus.com/products/temtop-m2000-air-quality-detector-co2-sensor-professional-hcho-co2-pm2-5-pm10-monitor. (дата звернення 18.03.2024).

19. uHoo Smart Air Monitor. URL: https://getuhoo.com/smart-air-monitor/. (дата звернення 18.03.2024).

20. Matthes E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. 3rd Edition. No Starch Press, January 10, 2023. 552 p.

21. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming. 2nd Edition. O'Reilly Media, May 10, 2022. 1012 p.

22. Parker J. Kotlin for Android development and Kotlin coroutines: A comprehensive guide to Android development for tech enthusiasts and beginners. March 28, 2024. 383 p.

23. Edet T. Android Development with Kotlin: Crafting Dynamic Mobile Experiences (Mobile App Development). CompreQuest Books, March 11, 2024. 269 p.

24. Smyth N. Android Studio Iguana Essentials - Java Edition: Developing Android Apps Using Android Studio 2023.2.1 and Java. Payload Media. March 18, 2024. 1277 p.

25. DeBarros A. Practical SQL: A Beginner's Guide to Storytelling with Data. 2nd Edition. No Starch Press, January 25, 2022. 464 p.

26. Gondosubroto R. Internet of Things from Scratch: Build IoT solutions for Industry 4.0 with ESP32, Raspberry Pi, and AWS. Packt Publishing, 2024. 438 p.

27. Dust Sensor. URL: https://www.waveshare.com/wiki/Dust_Sensor (дата звернення 20.03.2024).

28. Kumar A., Hussain J., Chun A. Connecting the Internet of Things: IoT Connectivity Standards and Solutions. Apress, 2023. 390 p.

29. Smeenk H. G. Internet of Things for Smart Buildings: Leverage IoT for smarter insights for buildings in the new and built environments. Packt Publishing, 2023. 306 p.

30. Halfacree G., Everard B. Get started with MicroPython on Raspberry Pi Pico: The Official Raspberry Pi Pico Guide. 2nd Edition. Raspberry Pi Press, August 6, 2024. 208 p.

31. Grinberg M., MicroPython for the Raspberry Pi Pico W: A gentle introduction to programming digital circuits with Python. October 29, 2022. 142 p.

32. Lancaster C. Air Pollution: Measurement, Modeling and Mitigation. States Academic Press, September 27, 2022. 242 p.

33. Moucha R. A. Internet of things (IoT). Journal of Data Analysis and Information Processing, 2021. P. 77-101.

34. Nolan A. Unlocking the Power of Raspberry Pi 5: Your Complete Guide from Setup to Expert Projects, 2024. 84 p.

35. Dian F. J. Fundamentals of Internet of Things: For Students and Professionals. John Wiley & Sons, 2022. 432 p.

36. Sonawane N. Learn Android Development - Vol. 1: Code your dreams into Android reality. March 8, 2024. 37 p.

37. BME680 Datasheet. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf (дата звернення 5.05.2024).

38. Low-cost outdoor air quality monitoring and sensor calibration: A survey and critical analysis / Concas F. et al. ACM Transactions on Sensor Networks (TOSN), 2021. pp. 1-44.

39. An Improvement Strategy for Indoor Air Quality Monitoring Systems / De Capua C. et al. Sensors, 2023.

40. Bell C. Beginning MicroPython with the Raspberry Pi Pico: Build Electronics and IoT Projects (Maker Innovations Series). Apress, July 24, 2022. 660 p.

41. MQ-135 Gas sensor technical data. URL: https://components101.com/sites/default/files/component_datasheet/MQ135%20Datasheet.pdf (дата звернення 05.04.2024).

42. MQ-135 Gas sensor. URL: https://www.waveshare.com/wiki/MQ-135_Gas_Sensor (дата звернення 05.04.2024).

43. Wambua M. S. Modern Android 13 Development Cookbook: Over 70 recipes to solve Android development issues and create better apps with Kotlin and Jetpack Compose. Packt Publishing. July 7, 2023. 322 p.

44. Boudjnah E. Clean Architecture for Android: Implement Expert-led Design Patterns to Build Scalable, Maintainable, and Testable Android Apps. BPB Publications, October 3, 2022. 262 p.

45. Dumbravan A. Clean Android Architecture: Take a layered approach to writing clean, testable, and decoupled Android applications. Packt Publishing, June 20, 2022. 368 p.

46. Efthymiou P. Clean Mobile Architecture: Become an Android, iOS, Flutter Architect. Petros Efthymiou, April 8, 2022. 302 p.

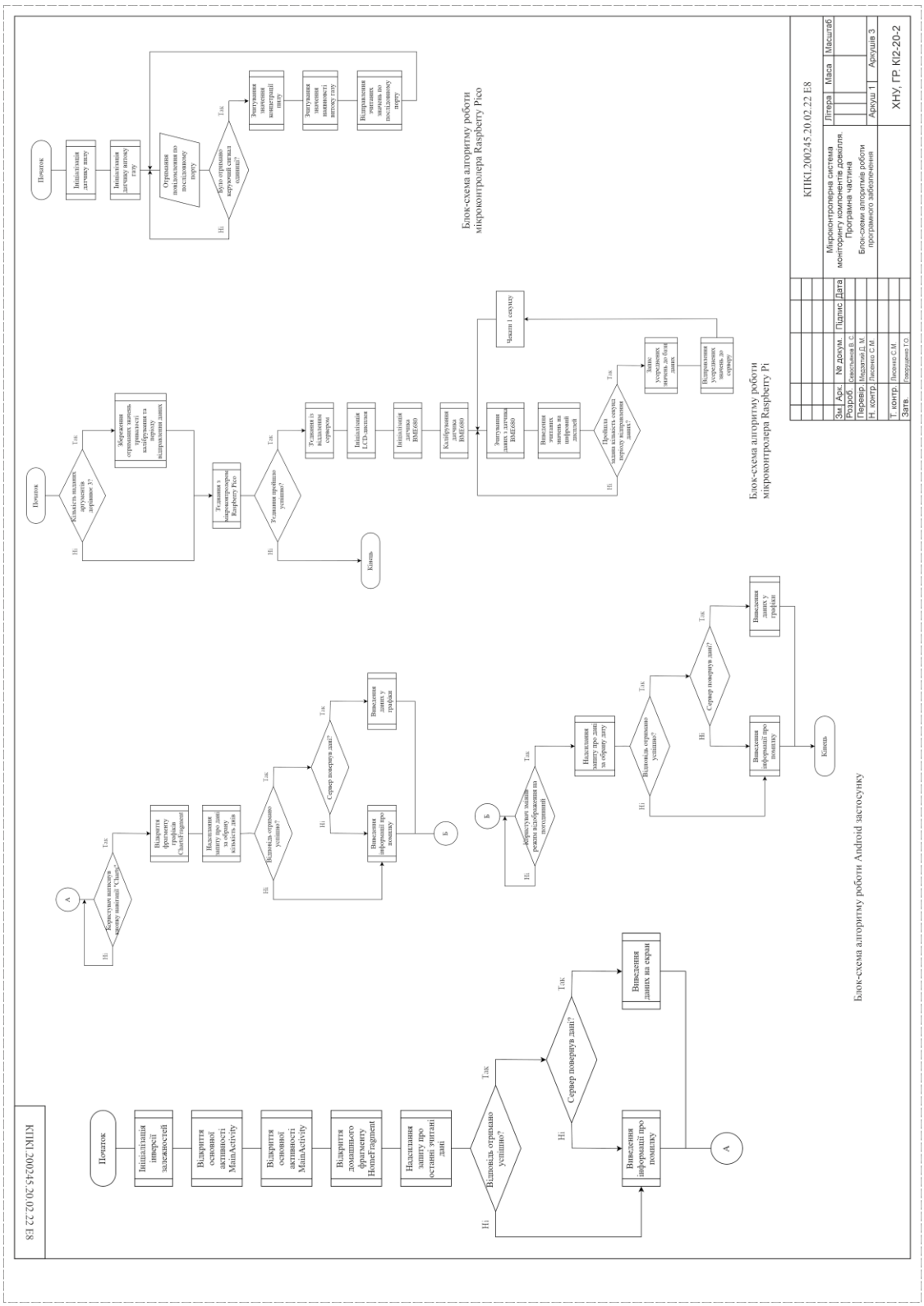
47. Hussain K., Hussain F. Kotlin Unleashed: Harnessing the Power of Modern Android Development Category. October 26, 2023. 324 p.

48. Kumar P. Mastering Android Studio: A Comprehensive Guide to Android App Development. Independently published, August 16, 2023. 53 p.

Додаток А

(обов’язковий)

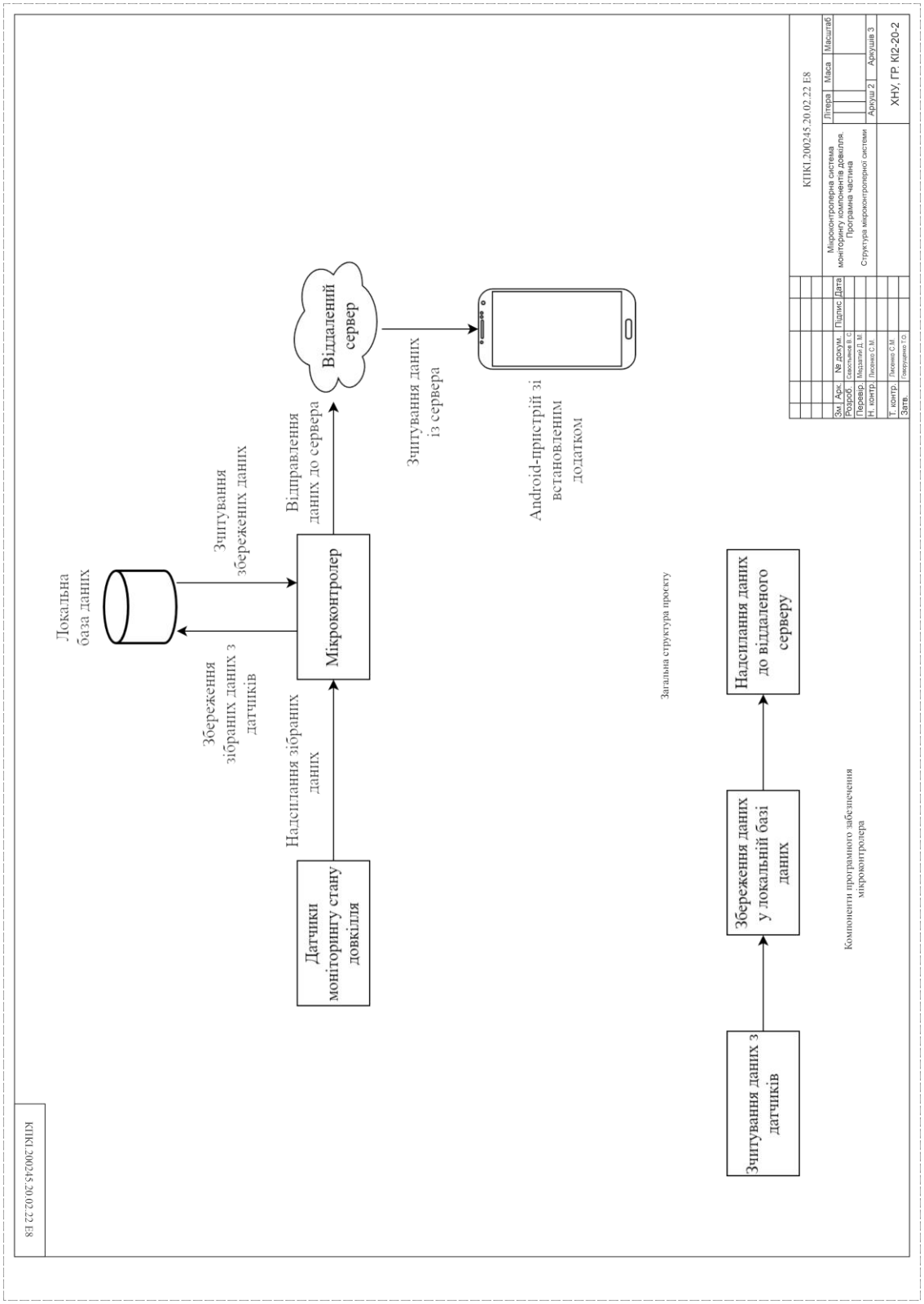
Копія креслення «Блок-схеми алгоритмів роботи програмного забезпечення»



Додаток Б

(обов’язковий)

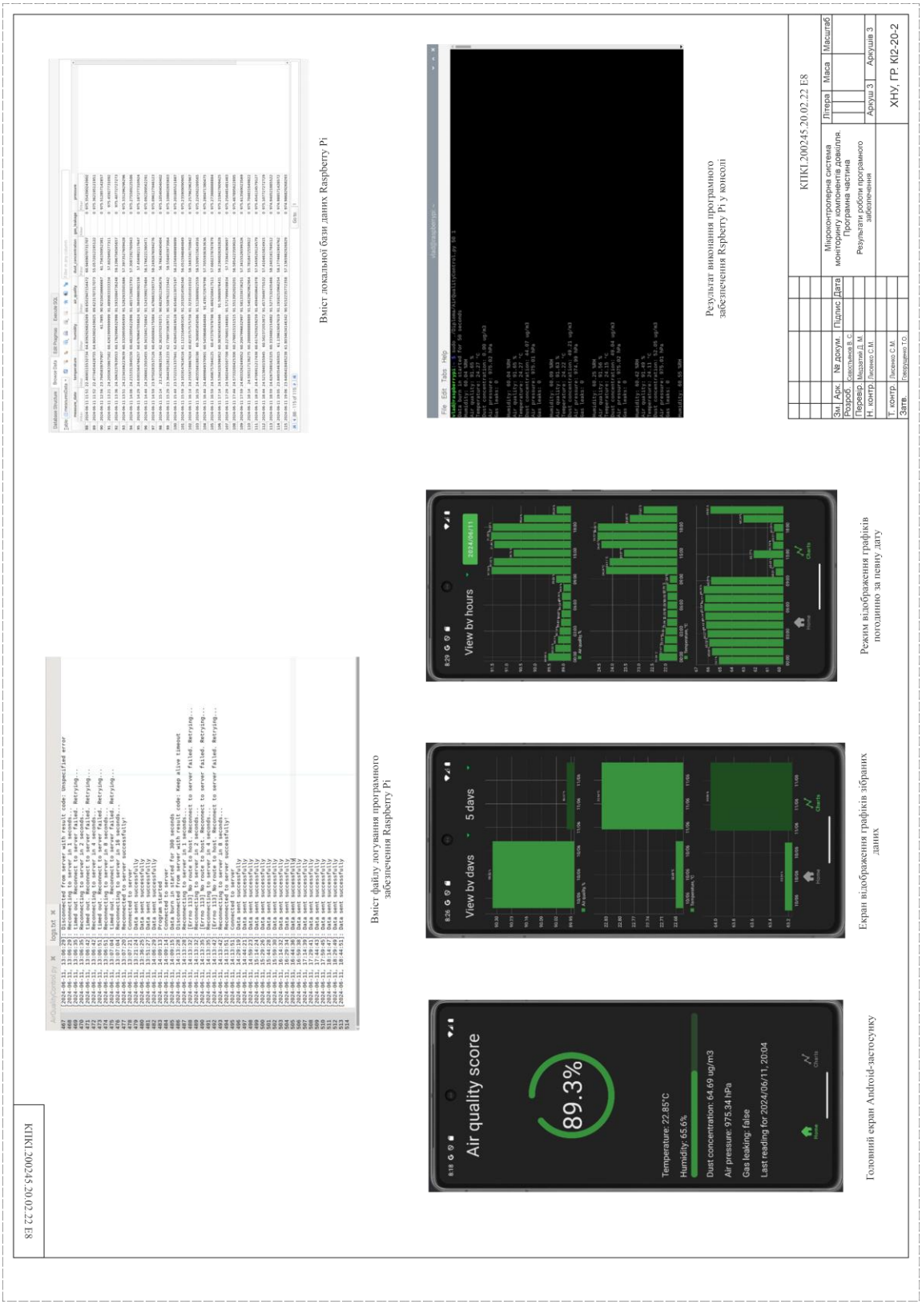
Копія креслення «Структура мікроконтролерної системи»



Додаток В

(обов’язковий)

Копія креслення «Результати роботи програмного забезпечення»



UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016368440

Дата перевірки:
17.06.2024 15:30:37 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
17.06.2024 16:45:20 EEST

ID користувача:
100005591

Назва документа: Сєвостьянов_Мікроконтролерна система моніторингу компонентів довкілля. Програмна ча...

Кількість сторінок: 65 **Кількість слів:** 8996 **Кількість символів:** 70833 **Розмір файлу:** 5.54 MB **ID файлу:** 1016175172

7.97%
Схожість

Найбільша схожість: 1.18% з джерелом з Бібліотеки (ID файлу: 1016162143)

7.31% Джерела з Інтернету	617	Сторінка 67
3.38% Джерела з Бібліотеки	180	Сторінка 70

0.37% Цитат

Цитати	4	Сторінка 71

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 131092 Назва: БКР Мікроконтролерна система моніторингу компонентів довкілля. Програмна частина Додано в БД: 2024-06-17 Автора: В. С. Сєвостьянов Керівники: Д. М. Медзатий Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	61476	538	1639 (3%)	18 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Євостьянов Владислав Сергійович

Тема: Мікроконтролерна система моніторингу компонентів довкілля. Програмна частина

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 57

1. Короткий зміст роботи та прийнятих рішень: У ході виконання дипломної роботи розроблено та імплементовано програмне забезпечення системи моніторингу компонентів довкілля.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи проаналізовано досліджувану проблему та існуючі рішення; побудовано план розробки алгоритмів роботи мікроконтролерної системи моніторингу компонентів довкілля. В другому розділі кваліфікаційної роботи розроблено загальні алгоритми роботи програмного забезпечення мікроконтролерної системи моніторингу компонентів довкілля. Спроектовано структуру системи, визначено її основні складові та описано загальні алгоритми роботи кожного з компонентів системи. В третьому розділі кваліфікаційної роботи деталізовано алгоритми роботи складових мікроконтролерної системи моніторингу компонентів довкілля. Розроблено програмне забезпечення кожної складової, що включає одноплатний комп'ютер, мікроконтролер та Android пристрій. Кожен із компонентів системи протестовано та продемонстровано результати роботи програмного забезпечення мікроконтролерної системи.

4. Позитивні сторони роботи: висока надійність розробленого програмного забезпечення та практична цінність роботи.

5. Негативні сторони роботи: незначні похибки при перетворенні зчитаних значень з датчиків системи.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

к.т.н., доцент кафедри АКІТтаР Корецька Л.О.

_____

"18" 06 2024 р. _____(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Мікроконтролерна система моніторингу компонентів довкілля. Програмна частина

Автор: Сєвостьянов Владислав Сергійович

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Медзатий Дмитро Миколайович, к.т.н., доц.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Виявлені в роботі запозичення не плагіатом, адже:

1) запозичення в описі досліджуваної проблеми використовують інформацію, що є історичними фактами, чи дослідженнями, які не стосуються результатів роботи;
2) запозичення в описі існуючих рішень використовують загальнодоступну інформацію про пристрої, що не є авторським дослідженням;
3) усі виявлені запозичення мають посилання на їх джерела;
4) виявлені запозичення є фрагментарними та не стосуються авторських досліджень;
5) в окремих місцях у якості запозичень було взято послідовність коефіцієнтів значення фільтру датчика BME680, що стосується лише опису роботи датчика та на результати виконаної роботи не впливають;
6) усі інші запозичення є комбінацією кількох слів, що загалом посилаються на вказані джерела, або загальновідомі факти, як от опис підтримуваних типів даних бібліотеки SQLite, чи типи сполук, що вловлюються сучасними системами моніторингу якості повітря.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 7.97% і адресується до 797 першоджерел, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Д. М. Медзатий

Гарант ОП _____ С.М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Сєвостьянова Владислава Сергійовича

ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-20-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

22 квітня 2024 року