Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

КЛАСТЕР ПРИКЛАДНИХ ПРОГРАМ З ВИКОРИСТАННЯМ МОБІЛЬНИХ ПРИСТРОЇВ ТА ІНТЕРНЕТУ РЕЧЕЙ

CLUSTER OF APPLICATIONS USING MOBILE DEVICES AND THE INTERNET OF THINGS

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІсз-21
Балас Павло Олександрович

(підпис)

Керівник:
к.е.н., доцент
Гордєєва Дар'я Валеріївна

(підпис)

Кваліфікаційну роботу
допущено до захисту
«_____» червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н. Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Баласу Павлу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Кластер прикладних програм з використанням мобільних пристроїв та інтернету речей*

Керівник роботи *к.е.н., доцент Гордєєва Дар'я Валеріївна*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Теоретичні основи
Проєктування системи
Реалізація та дослідження роботи системи
Висновки

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Теоретичний огляд	*Гордєєва Д.В., доцент*		
Вибір засобів та методів	*Гордєєва Д.В., доцент*		
Розробка геймпад пістолета	*Гордєєва Д.В., доцент*		
Показник запозичень тексту	***%***		
Академічна доброчесність	*Міскевич О.І., асистент*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми*	До 15.11.2022 р.	
2.	*Розділ 1. Теоретичні основи*	До 15.12.2022 р.	
3.	*Розділ 2. Проєктування системи*	До 02.02.2023 р.	
4.	*Розділ 3. Реалізація та дослідження роботи системи*	До 02.04.2023 р.	
5.	*Висновки та пропозиції*	До 15.04.2023 р.	
6.	*Формування списку використаних джерел*	До 02.05.2023 р.	
7.	*Оформлення ілюстративного матеріалу*	До 15.05.2023 р.	
8.	*Нормоконтроль*	До 25.05.2023 р.	
9.	*Інструментальна перевірка на академічний плагіат*	До 01.06.2023 р.	
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	До 07.06.2023 р.	

Здобувач вищої освіти _____ Балас П.О.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Гордєєва Д.В.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Балас П.О. Кластер прикладних програм з використанням мобільних пристроїв та інтернету речей. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023. 59 с.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел.

Перший розділ присвячено огляду предметної області моніторингу параметрів мікроклімату приміщень, розглянуто базові поняття та архітектурні підходи до побудови IoT-систем, а також особливості використання зв'язки «ESP32 – хмарні сервіси – мобільний клієнт» для розподілених вимірювальних комплексів.

У другому розділі здійснено вибір та обґрунтування засобів розробки системи: обрано стек ESP32 + Firebase + Android, спроєктовано загальну архітектуру кластера прикладних програм, структуру даних у Firebase, а також розподіл функцій між мікроконтролером, хмарною базою та мобільним застосунком.

Третій розділ присвячено безпосередній реалізації системи моніторингу мікроклімату: налаштуванню середовищ розробки, програмній реалізації прошивки ESP32 та Android-застосунку, конфігурації Firebase, побудові екрану з графічною візуалізацією історії вимірювань і проведенню інтеграційних експериментальних досліджень.

Ключові слова: контролер ESP32, Firebase, Android, Інтернет речей, моніторинг мікроклімату, мобільний застосунок.

ANNOTATION

Balas P. Cluster of applications using mobile devices and the Internet of Things. Manuscript.

Qualifying work of a bachelor of EP «Computer Engineering» specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023. 59 p.

The qualification work consists of an introduction, three sections, conclusions, and a list of used sources.

The first chapter is devoted to an overview of the problem domain of indoor microclimate monitoring. It considers the basic concepts and architectural approaches to building IoT systems, as well as the specifics of using the “ESP32 – cloud services – mobile client” stack for distributed measurement complexes.

The second chapter focuses on the selection and justification of the development tools: the ESP32 + Firebase + Android stack is chosen, the overall architecture of the application cluster is designed, along with the data structure in Firebase and the distribution of responsibilities between the microcontroller, the cloud database and the mobile application.

The third chapter is dedicated to the practical implementation of the microclimate monitoring system: configuration of development environments, software implementation of the ESP32 firmware and the Android application, Firebase configuration, development of a screen with graphical visualization of measurement history, and conducting integrated experimental studies.

Keywords: ESP32 microcontroller, Firebase, Android, Internet of Things, microclimate monitoring, mobile application.

ЗМІСТ

ВСТУП

Стрімкий розвиток інформаційних технологій, поширення мобільних пристроїв та концепції Інтернету речей зумовлюють необхідність переходу від ізольованих програмних продуктів до інтегрованих, розподілених рішень, здатних функціонувати як єдине цілісне середовище. У сучасних умовах користувач очікує безперервного доступу до даних у режимі реального часу, незалежно від того, де саме ці дані було згенеровано: на сенсорі в приміщенні, на мікроконтролері у вбудованій системі чи у хмарному сервісі. Це особливо актуально для задач моніторингу параметрів мікроклімату, контролю споживання ресурсів, побудови інтелектуальних систем керування житловими та виробничими приміщеннями, де поєднуються фізичні пристрої, мобільні застосунки та хмарна інфраструктура.

Інтернет речей, мобільні платформи та хмарні сервіси формують технологічний фундамент для побудови кластеризованих програмних рішень, у яких окремі прикладні програми не лише виконують власні локальні функції, а й взаємодіють між собою через уніфіковане інформаційне середовище. У такому підході мікроконтролер, що зчитує показники сенсорів, мобільний застосунок, що візуалізує ці показники для користувача, та хмарний сервіс, що забезпечує зберігання, обробку й синхронізацію даних, розглядаються як взаємопов’язані компоненти єдиного кластера прикладних програм. Це дозволяє підвищити гнучкість, масштабованість та відмовостійкість системи, а також забезпечити можливість її подальшого розвитку без радикальних змін архітектури.

Актуальність обраної теми зумовлена тим, що в навчальній та практичній діяльності майбутніх фахівців з комп’ютерної інженерії особливого значення набуває вміння створювати комплексні рішення, які інтегрують апаратні компоненти, мобільні клієнтські застосунки та хмарну інфраструктуру. Розроблення кластера прикладних програм на основі мікроконтролера ESP32, хмарної платформи типу Backend-as-a-Service для зберігання та обміну даними та мобільного застосунку для операційної системи Android дає змогу на практиці

продемонструвати повний цикл створення сучасної кіберфізичної системи, орієнтованої на взаємодію з кінцевим користувачем.

Метою кваліфікаційної роботи є розроблення та дослідження кластера прикладних програм з використанням мобільних пристроїв та технологій Інтернету речей, реалізованого на основі мікроконтролера ESP32, хмарної бази даних Firebase та мобільного застосунку для платформи Android. Досягнення поставленої мети передбачає послідовне розв’язання завдань, пов’язаних з аналізом теоретичних засад побудови мобільно-IoT систем, дослідженням можливостей інтеграції апаратного, програмного та хмарного рівнів, проєктуванням архітектури кластера прикладних програм, реалізацією його компонентів та оцінюванням функціональних і експлуатаційних характеристик створеної системи.

Об’єктом дослідження в роботі виступають процеси побудови та функціонування розподілених програмно-апаратних систем, що поєднують мобільні пристрої, мікроконтролери Інтернету речей та хмарні сервіси зберігання даних. Предметом дослідження є методи й програмні засоби організації кластера прикладних програм, які забезпечують інтеграцію мікроконтролера ESP32, хмарної платформи Firebase та мобільного Android-застосунку в єдине інформаційне середовище з підтримкою обміну даними в режимі, наближеному до реального часу.

Наукова новизна роботи полягає в тому, що кластер прикладних програм розглядається не лише як набір окремих застосунків, а як цілісна програмно-апаратна конфігурація, у якій за рахунок використання хмарної платформи типу Firebase забезпечується логічна єдність даних, прозора взаємодія між компонентами та можливість подальшого масштабування системи. Практична цінність полягає у створенні працездатного прототипу системи моніторингу параметрів середовища, який може бути використаний як навчальний стенд, основа для подальших досліджень або як базова платформа для побудови реальних прикладних рішень у сфері Інтернету речей.

Завдання, які необхідно виконати:

– проаналізувати основні принципи побудови IoT-систем та можливості використання ESP32, Firebase і Android для моніторингу мікроклімату;

– обґрунтувати вибір стеку технологій ESP32 + Firebase + Android для реалізації системи моніторингу приміщення;

– спроєктувати архітектуру системи, структуру даних у Firebase та взаємодію між ESP32, хмарою і мобільним застосунком;

– розробити та реалізувати програмне забезпечення: прошивку для ESP32 та мобільний застосунок Android з відображенням поточних і історичних даних;

– провести тестування роботи системи в реальних умовах, оцінити її коректність, швидкодію та зручність використання, сформулювати висновки й рекомендації.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ МОБІЛЬНО-IoT СИСТЕМИ

1.1 Поняття та архітектура Інтернету речей

Інтернет речей розглядається як еволюційний етап розвитку глобальної мережі, на якому до традиційних обчислювальних пристроїв приєднується велика кількість фізичних об’єктів, оснащених вбудованими засобами вимірювання, обробки та передавання даних. На відміну від класичної моделі Інтернету, де домінують персональні комп’ютери, сервери та мобільні термінали, в Інтернеті речей ключовим учасником інформаційного обміну стають сенсорні вузли, виконавчі пристрої, побутова техніка, елементи інфраструктури будівель та виробничого обладнання. Ці об’єкти набувають статусу логічних «вузлів» мережі, здатних автономно генерувати, передавати та споживати дані, а також реагувати на події у фізичному та інформаційному середовищах.

З формальної точки зору Інтернет речей можна описати як багаторівневу систему, у якій виділяють рівень фізичних пристроїв, мережевий рівень, платформний або хмарний рівень та рівень прикладних застосунків. На рівні фізичних пристроїв функціонують мікроконтролери та сенсори, що безпосередньо взаємодіють з навколишнім середовищем. Вони вимірюють параметри, такі як температура, вологість, освітленість, тиск, вібрації, та перетворюють ці значення у цифрову форму. Прикладом такого вузла є мікроконтролер із вбудованим модулем бездротового зв’язку, здатний одночасно виконувати базову обчислювальну логіку, керувати підключеними сенсорами та передавати зібрані дані до мережі.

Мережевий рівень відповідає за маршрутизацію й транспортування даних між IoT-пристроями та іншими компонентами системи. У цьому контексті застосовуються як традиційні протоколи стеку TCP/IP, так і спеціалізовані протоколи прикладного рівня, оптимізовані для середовищ із обмеженими ресурсами та ненадійними каналами зв’язку. Для реалізації зв’язку між мікроконтролером і хмарною платформою можуть використовуватися захищені

HTTP-запити, легковагові протоколи обміну повідомленнями або їх комбінації. Важливою особливістю є те, що мережевий рівень в Інтернеті речей повинен враховувати енергетичні обмеження, низьку пропускну здатність та потенційно високий рівень втрат пакетів.

На платформному, або хмарному, рівні зосереджено функції зберігання, агрегування та обробки даних, а також організації доступу до них. Хмарні платформи, орієнтовані на підтримку IoT-сценаріїв, забезпечують можливість прийому даних від великої кількості розподілених пристроїв, їх структурування, довготривалого зберігання та подальшої аналітики. У межах спрощеного підходу, що розглядається в даній роботі, роль такого середовища виконує платформа Firebase, яка поєднує функції хмарної бази даних, сервісів автентифікації та механізмів обміну даними в режимі, наближеному до реального часу. Використання готової інфраструктури типу Backend-as-a-Service дозволяє зменшити обсяг серверного програмування, зосередивши основні зусилля на інтеграції апаратної частини та клієнтських застосунків.

Рівень прикладних застосунків в Інтернеті речей є тим шаром, де реалізується безпосередня взаємодія з користувачем, відбувається інтерпретація даних та прийняття рішень. У сучасних системах особливо важливу роль відіграють мобільні застосунки, що забезпечують зручний доступ до інформації й засобів керування у будь-який час та з будь-якого місця. Мобільна платформа Android у цьому контексті виступає природним вибором завдяки поширеності відповідних пристроїв, розвиненій екосистемі інструментів розробки та наявності офіційних засобів інтеграції з хмарними сервісами, зокрема Firebase. Саме мобільний додаток на Android у запропонованому рішенні виконує функції візуалізації даних IoT-пристрою, конфігурування параметрів роботи системи та формування керуючих впливів.

Отже, архітектура Інтернету речей у загальному випадку відображає розподіл функцій між фізичним, мережевим, хмарним та прикладним рівнями. У контексті даної роботи ця архітектура конкретизується у вигляді зв’язки «ESP32 – Firebase – Android», де мікроконтролер ESP32 реалізує сенсорний та виконавчий рівень, Firebase – хмарний рівень зберігання й синхронізації даних,

а мобільний застосунок Android – рівень користувацької взаємодії. Така трикомпонентна схема дозволяє втілити концепцію кластеризованої системи, у якій окремі прикладні програми, розгорнуті на різних платформах, працюють як взаємопов’язаний програмно-апаратний комплекс.

1.2 Мікроконтролер ESP32 як платформа для IoT-рішень

Мікроконтролер ESP32 належить до класу вбудованих систем із інтегрованими засобами бездротового зв’язку і є однією з найпоширеніших апаратних платформ для створення доступних за вартістю, але функціонально потужних IoT-рішень. На відміну від класичних мікроконтролерів, що потребують додаткових зовнішніх модулів для підключення до мережі, ESP32 має вбудований модуль Wi-Fi, а в більшості модифікацій також і Bluetooth, що істотно спрощує його інтеграцію до бездротових інфраструктур. Наявність двоядерного процесора, достатнього обсягу оперативної пам’яті та флеш-пам’яті дає змогу реалізовувати на ESP32 не лише базову логіку зчитування сенсорів, а й початкову обробку даних, фільтрацію, усереднення та формування структурованих повідомлень для передавання в хмарне середовище.

З погляду розробника, ESP32 є привабливою платформою завдяки підтримці популярних інструментів програмування та наявності розвинутої екосистеми бібліотек. Для освітніх та прототипних задач найчастіше використовується середовище Arduino IDE, яке дозволяє програмувати ESP32 мовою C/C++ із застосуванням стандартних функцій та знайомих бібліотек. Такий підхід знижує поріг входу, оскільки значна частина навчальних матеріалів і прикладів для класичних плат Arduino може бути адаптована для ESP32 з мінімальними змінами. Водночас у більш складних проєктах можливе використання альтернативних середовищ, таких як PlatformIO або фреймворк ESP-IDF, що відкриває доступ до низькорівневих можливостей контролера та розширених засобів налагодження.

З технічної точки зору ESP32 забезпечує широкий спектр апаратних інтерфейсів, необхідних для підключення сенсорів і виконавчих пристроїв. До

складу контролера входять цифрові входи й виходи, аналогово-цифрові перетворювачі, інтерфейси I²C, SPI, UART, PWM-канали для керування двигунами або яскравістю світлодіодів. Це дає змогу будувати на основі ESP32 як прості сенсорні вузли для вимірювання температури та вологості повітря, так і складніші системи з кількома типами датчиків та елементами керування. У контексті даної роботи ESP32 використовується для створення вузла моніторингу параметрів середовища, який у реальному часі зчитує значення з підключених сенсорів, наприклад температури та відносної вологості, та передає сформовані дані до хмарної бази Firebase.

Фундаментальною вимогою до IoT-платформи є здатність підтримувати сталий зв’язок із хмарною інфраструктурою та коректно обробляти мережеві збої. ESP32 має програмні стекові рішення для роботи з протоколами TCP/IP, що дозволяє встановлювати захищені HTTP-з’єднання з хмарними сервісами. У найпростішому сценарії, який відповідає обраному стеку технологій, прошивка ESP32 періодично ініціює підключення до хмарної бази даних Firebase через HTTPS, формує HTTP-запити з JSON-представленням виміряних даних і надсилає їх на відповідну адресу. Такий підхід дозволяє використовувати Firebase як єдину точку зберігання даних без потреби у розгортанні власного серверного програмного забезпечення.

Окрім відправлення даних, ESP32 може виступати клієнтом, який зчитує керуючу інформацію з хмарної бази. Це створює передумови для реалізації двосторонньої взаємодії між фізичним пристроєм та мобільним застосунком через посередництво Firebase. Наприклад, користувач, працюючи з Android-застосунком, змінює значення певного параметра у хмарі (порогову температуру, режим роботи чи стан віртуального перемикача), після чого ESP32 періодично звертається до відповідного вузла в базі даних та застосовує отримане значення до своєї логіки керування. У такій моделі мікроконтролер виступає не лише джерелом даних для хмари, а й виконавцем рішень, ініційованих користувачем через мобільний інтерфейс.

З огляду на навчальні та дослідницькі завдання, ESP32 має ще одну важливу перевагу, а саме доступність великої кількості прикладів, документації

та спільнот підтримки. Це значно спрощує пошук рішень типових проблем, пов’язаних із налаштуванням мережевого підключення, інтеграцією з популярними сенсорами чи хмарними сервісами. Для бакалаврської кваліфікаційної роботи це означає можливість зосередитися не стільки на подоланні низькорівневих технічних складностей, скільки на проєктуванні архітектури системи та дослідженні її властивостей як елемента кластеризованого програмно-апаратного комплексу.

Таким чином, мікроконтролер ESP32 у межах даної роботи розглядається як базова апаратна платформа для реалізації вузла Інтернету речей, що поєднує функції збирання, первинної обробки та передавання даних до хмарного середовища, а також приймання керуючих команд. У зв’язці з Firebase та мобільним застосунком на Android він формує нижній рівень кластера прикладних програм, забезпечуючи фізичний зв’язок розробленої системи з навколишнім середовищем і створюючи основу для подальшої інтеграції й масштабування мобільно-IoT рішень.

1.3 Хмарна платформа Firebase та її можливості для зберігання й обміну даними

Розвиток хмарних технологій суттєво змінив підходи до проєктування розподілених інформаційних систем, зокрема систем Інтернету речей. Замість розгортання власної серверної інфраструктури розробники можуть використовувати сервіси типу Backend-as-a-Service, які надають уніфіковані інтерфейси для зберігання даних, автентифікації користувачів, реалізації обміну повідомленнями та виконання серверної логіки. Однією з таких платформ є Firebase, що поєднує низку сервісів, орієнтованих на мобільні та веб-застосунки, та добре інтегрується з концепцією IoT у випадках, коли потрібна проста й водночас гнучка хмарна складова.

З погляду даної роботи ключове значення мають сервіси Firebase Realtime Database або Cloud Firestore, які забезпечують структурування та зберігання даних у хмарі у формі, зручною для клієнтів, що працюють у режимі реального

часу. Обидві технології використовують модель документо- або деревоорієнтованих баз даних, де інформація організована у вигляді ієрархії вузлів або колекцій документів, до яких можна звертатися через уніфіковані програмні інтерфейси. Для системи «ESP32 – Firebase – Android» це означає, що вимірювання, які надсилає мікроконтролер, можуть бути безпосередньо записані у певну гілку бази даних, а мобільний застосунок отримує повідомлення про зміну цих даних практично миттєво, без потреби опитувати сервер із фіксованими інтервалами.

Важливою особливістю Firebase є те, що платформа сама бере на себе задачі підтримання постійного з’єднання з клієнтом, обробки змін на рівні даних та трансляції цих змін на всі підключені клієнтські застосунки. У випадку Android-додатка це виявляється у можливості підписатися на певний вузол або колекцію даних та автоматично отримувати події про оновлення, додавання чи видалення записів. Таким чином, Firebase виступає не лише пасивним сховищем інформації, а й активним посередником у процесі синхронізації станів між компонентами кластера прикладних програм. Для розробника це означає, що значна частина складних нижньорівневих механізмів, пов’язаних із підтриманням з’єднань і розповсюдженням подій, прихована за високорівневими бібліотеками.

Ще одним суттєвим аспектом є система правил безпеки Firebase, яка дозволяє гнучко визначати права доступу до різних частин бази даних. За допомогою декларативних політик можна обмежити можливість читання чи запису даних лише авторизованими клієнтами, ввести перевірку належності ресурсу конкретному користувачеві або задати інші умови доступу. У контексті системи, де ESP32 надсилає дані з сенсорів, а Android-застосунок їх відображає та формує керуючі команди, правила безпеки дозволяють, з одного боку, захистити від несанкціонованого втручання, а з іншого – організувати керовану взаємодію між різними учасниками системи. При цьому реалізація базових сценаріїв автентифікації (наприклад, за допомогою електронної пошти та пароля або анонімного входу) зводиться до використання готових компонентів Firebase Authentication.

Firebase також може виступати середовищем виконання частини серверної логіки за допомогою механізмів хмарних функцій. Хоча в межах спрощеного стеку, обраного для цієї кваліфікаційної роботи, основний акцент робиться на прямій взаємодії «ESP32 – база даних – Android», платформу можна розширити за рахунок запуску функцій, які автоматично реагують на зміну певних вузлів у базі даних. Це відкриває можливості для реалізації додаткових сервісів, таких як формування push-сповіщень при виході параметрів за встановлені межі, виконання періодичної агрегації даних або збереження архівів вимірювань в окремі сховища.

Таким чином, Firebase у зв’язці з ESP32 та Android виконує роль центральної хмарної ланки, яка об’єднує апаратний рівень та клієнтський інтерфейс у єдине інформаційне середовище. Завдяки підтримці обміну в режимі, наближеному до реального часу, наявності механізмів безпеки та простим засобам інтеграції з мобільними застосунками Firebase дозволяє розглядати кластер прикладних програм як логічно цілісну систему, незважаючи на фізичний розподіл її компонентів у мережі.

1.4 Мобільна платформа Android як клієнтський інтерфейс IoT-систем

Мобільна платформа Android посідає провідне місце на ринку смарт-пристроїв, що робить її природним вибором для створення клієнтських застосунків до систем Інтернету речей. Висока поширеність Android-смартфонів і планшетів, розвинена екосистема інструментів розробки та наявність офіційних бібліотек для роботи з хмарними сервісами, зокрема Firebase, дозволяють реалізувати зручний, інтуїтивно зрозумілий інтерфейс доступу до даних і функцій IoT-системи для широкого кола користувачів. У зв’язці «ESP32 – Firebase – Android» саме мобільний застосунок виступає тим елементом, через який користувач сприймає роботу всієї системи та взаємодіє з нею.

Архітектура Android-застосунків побудована на ідеї компонентного підходу, де основними структурними елементами є активності, фрагменти, служби та інші допоміжні компоненти. Така модель дозволяє логічно розділити

інтерфейс користувача, бізнес-логіку та фонові процеси, що особливо важливо для систем, які працюють з даними в реальному часі. У типовому сценарії для IoT-системи головна активність або набір фрагментів відповідають за відображення поточних значень параметрів, історії вимірювань та засобів керування, тоді як взаємодія з Firebase виноситься у спеціалізовані класи або служби, що працюють у тісній взаємодії з життєвим циклом застосунку.

Офіційний SDK Firebase для Android забезпечує розробника готовими засобами підключення до хмарної бази даних, автентифікації користувачів та обробки подій зміни даних. Це означає, що в межах клієнтського коду немає потреби реалізовувати низькорівневі протоколи зв’язку, організовувати власні механізми підтримання постійних з’єднань або розробляти складні структури обробки подій. Достатньо підключити відповідні бібліотеки, описати структуру об’єктів, що відображають дані Firebase у внутрішніх моделях застосунку, та зареєструвати обробники змін, щоб отримувати актуальні значення параметрів майже миттєво після їхнього оновлення мікроконтролером ESP32.

Інтеграція Android-застосунку з Firebase не обмежується лише читанням даних. Користувач може здійснювати керувальні дії, які втілюються у вигляді змін певних записів у базі даних, а далі інтерпретуються на стороні ESP32 як команди. Наприклад, зміна логічного стану віртуального перемикача у мобільному інтерфейсі призводить до оновлення відповідного поля у Firebase. Мікроконтролер, періодично опитуючи це поле або використовуючи механізми сповіщення, сприймає зміну як інструкцію для увімкнення чи вимкнення виконавчого пристрою. У такій моделі Android-додаток не підключається безпосередньо до IoT-пристрою, але забезпечує інтуїтивний та безпечний механізм керування через хмарне посередництво.

Окремої уваги потребує питання візуалізації даних у мобільному інтерфейсі. Для користувача важливо не лише бачити поточні значення, а й мати можливість аналізувати динаміку зміни параметрів у часі. Це вимагає реалізації графічних компонентів, здатних відображати часові ряди, індикатори перевищення порогових значень, попереджувальні повідомлення тощо. Android надає широкий спектр можливостей для створення таких елементів, як власними

засобами, так і за допомогою сторонніх бібліотек для побудови графіків. У контексті обраного стеку технологій це дозволяє перетворити сирі вимірювання, що надходять із ESP32 через Firebase, на наочні візуальні образи, зрозумілі кінцевому користувачу.

Особливістю мобільних платформ є обмеження обчислювальних ресурсів, енергоспоживання та надійності мережевого з’єднання. Тому при розробленні Android-застосунку для IoT-системи необхідно враховувати оптимальне використання ресурсів, мінімізацію зайвої мережевої активності, а також коректне опрацювання ситуацій втрати зв’язку з хмарою. Firebase певною мірою полегшує ці задачі, надаючи механізми кешування даних на клієнті та синхронізації їх при відновленні з’єднання, однак правильне проєктування логіки застосунку залишається відповідальністю розробника.

У підсумку мобільна платформа Android у системі «ESP32 – Firebase – Android» виконує роль універсального клієнтського інтерфейсу, який забезпечує людині доступ до даних і функцій IoT-системи, реалізує візуалізацію стану середовища та надає інструменти для керування пристроями. Завдяки тісній інтеграції з Firebase та широким можливостям для побудови інтерфейсів Android-додаток стає повноцінним елементом кластера прикладних програм, у якому апаратний, хмарний та клієнтський рівні працюють як узгоджена програмно-апаратна система.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ «ESP32 – FIREBASE – ANDROID»

2.1 Характеристика предметної області та постановка задачі

У межах даної кваліфікаційної роботи предметною областю обрано задачу моніторингу параметрів мікроклімату в приміщенні. Такий вибір є виправданим як з точки зору практичної значущості, так і з огляду на технічну реалізовуваність із використанням обраного стеку технологій. У навчальних аудиторіях, лабораторіях, офісних приміщеннях та житлових кімнатах параметри мікроклімату, насамперед температура та відносна вологість повітря, безпосередньо впливають на комфорт перебування людей, працездатність обладнання та енергоефективність систем опалення й вентиляції. Відсутність оперативної інформації про стан середовища ускладнює прийняття обґрунтованих рішень щодо режимів опалення, провітрювання чи використання кондиціонування, що може призводити до надлишкових витрат ресурсів або погіршення умов перебування.

Традиційні засоби контролю мікроклімату здебільшого обмежуються автономними термометрами чи гігрометрами, показники яких потрібно зчитувати безпосередньо на місці встановлення. Такий підхід не забезпечує централізованого збирання даних, накопичення історії вимірювань та їх аналізу в часі, а також не дозволяє організувати дистанційний доступ до інформації й реалізувати навіть елементарні сценарії автоматизованого реагування. Натомість використання системи Інтернету речей дає змогу побудувати розподілену систему моніторингу, у якій вимірювальні вузли передають дані у хмарне середовище, а користувач одержує доступ до актуальних і архівних значень через мобільний застосунок.

Узагальнену структурну схему системи моніторингу параметрів мікроклімату приміщення на основі зв’язки «ESP32 – Firebase – Android» подано на рисунку 2.1. На ній виділено три основні рівні: сенсорний рівень, представлений датчиком температури та відносної вологості, підключеним до мікроконтролера ESP32; хмарний рівень, який реалізовано у вигляді бази даних

та сервісів платформи Firebase; та клієнтський рівень, що відповідає мобільному застосунку на пристрої під керуванням операційної системи Android. Мікроконтролер ESP32 періодично зчитує показники сенсора, виконує первинну обробку отриманих значень і надсилає сформовані дані до Firebase через бездротове Wi-Fi-з'єднання. Мобільний застосунок, у свою чергу, підключається до тієї ж хмарної бази, одержує актуальні вимірювання в режимі, наближеному до реального часу, та відображає їх користувачеві у зручній формі.

Рисунок 2.1 – Структурна схема системи моніторингу параметрів мікроклімату приміщення на основі ESP32, Firebase та Android

Окрім односпрямованого потоку даних від сенсорного вузла до користувача, система передбачає можливість зворотнього впливу. Користувач, змінюючи налаштування у мобільному застосунку, може модифікувати відповідні записи у Firebase, наприклад коригувати порогові значення або режим опитування сенсорів. Ці зміни стають доступними для ESP32, який періодично зчитує конфігураційні параметри з хмарної бази та адаптує власну логіку роботи. Таким чином, реалізується двосторонній обмін інформацією, у якому Firebase виступає центральною ланкою між фізичним пристроєм та мобільним клієнтом.

З формальної точки зору поставлену задачу можна сформулювати як проєктування та реалізацію розподіленої програмно-апаратної системи, що складається з одного або кількох вимірювальних вузлів на базі ESP32, хмарної бази даних Firebase та мобільного клієнта Android, об’єднаних у логічно цілісний кластер прикладних програм. На рівні вимірювального вузла необхідно забезпечити стабільне зчитування показників із сенсора, перетворення їх у зручний для передавання формат та відправлення до хмарної бази з заданою періодичністю. На хмарному рівні слід спроєктувати структуру даних, що дозволяє зберігати вимірювання з прив’язкою до часу та вузла, а також організувати механізми контролю доступу. На рівні мобільного застосунку потрібно реалізувати функції отримання та візуалізації даних, взаємодії з Firebase у режимі, наближеному до реального часу, та формування керуючих впливів у межах обраного сценарію використання.

Постановка задачі також включає вимоги до якості функціонування системи. Зокрема, кластер прикладних програм має забезпечувати достатню актуальність даних для прийняття рішень щодо стану мікроклімату, що означає обмеження на максимально допустиму затримку між фактичним вимірюванням і відображенням значення у мобільному застосунку. Важливим є також забезпечення надійності роботи в умовах можливих втрат мережевого з’єднання, коректної поведінки при тимчасовій недоступності хмарного сервісу та можливості відновлення передавання даних без втрати цілісності історії вимірювань. Окремо слід врахувати аспекти безпеки, пов’язані з обмеженням несанкціонованого доступу до даних мікроклімату певного приміщення та запобіганням неавторизованому втручанню в параметри роботи системи.

2.2 Загальна архітектура системи та розподіл функцій між ESP32, Firebase та Android

Загальна архітектура системи моніторингу мікроклімату базується на трирівневій моделі, у якій чітко розмежовано функції апаратного вимірювального вузла, хмарної інфраструктури та клієнтського мобільного

застосунку. Такий підхід дозволяє спроєктувати кластер прикладних програм як сукупність взаємопов’язаних компонентів, кожен з яких виконує власну локальну задачу, але водночас забезпечує узгоджену роботу всієї системи в цілому. На нижньому рівні архітектури розташовано вимірювальний вузол на базі мікроконтролера ESP32 з підключеним датчиком температури та відносної вологості повітря. Середній рівень становить хмарна платформа Firebase, що відповідає за довготривале зберігання, структуризацію та синхронізацію даних. Верхній рівень представлено мобільним застосунком для Android, який виконує роль користувацького інтерфейсу, через який здійснюється доступ до інформації та керування параметрами роботи системи.

На рівні ESP32 реалізується безперервний цикл збирання та первинної обробки даних. Мікроконтролер з фіксованим часовим інтервалом ініціює опитування підключеного сенсора, перетворює отримані сирі значення на фізично осмислені параметри, такі як температура у градусах Цельсія та відносна вологість у відсотках, та формує зіставні між собою записи. Далі ці дані упаковуються у структуру, придатну для передавання через мережу у форматі JSON, доповнюються часовою міткою та, за потреби, ідентифікатором вузла. Після цього ESP32 встановлює захищене з’єднання з хмарою та надсилає сформований пакет у Firebase. Важливо, що на цьому ж рівні може виконуватися елементарна логіка контролю, наприклад відкидання явно некоректних показників або обмеження частоти надсилання при стабільних умовах, що зменшує навантаження на мережу та хмарну базу.

На хмарному рівні дані, отримані від ESP32, приймаються та зберігаються у структурі, яка забезпечує прив’язку кожного вимірювання до часу та конкретного пристрою. Firebase виконує функцію центрального сховища, у якому накопичуються усі записи, необхідні як для відображення поточного стану мікроклімату, так і для подальшого аналізу історичних трендів. Крім того, саме на цьому рівні відбувається логічне «розведення» потоків даних між різними клієнтами: одна й та сама інформація може одночасно використовуватися кількома мобільними застосунками, веб-інтерфейсами чи додатковими аналітичними сервісами без необхідності змінювати прошивку вимірювальних

вузлів. Firebase також забезпечує механізми контролю доступу та автентифікації, завдяки чому розмежовується доступ до даних різних користувачів і запобігається несанкціонованому втручанню.

Мобільний застосунок Android, який становить верхній рівень архітектури, підключається до Firebase як клієнт, що споживає дані, та одночасно як джерело керуючих впливів. Завдяки механізмам підписки на зміни, які надає платформа, застосунок одержує актуальну інформацію про параметри мікроклімату практично відразу після її оновлення у хмарі, незалежно від фізичного розташування пристрою. На рівні інтерфейсу користувача ці дані візуалізуються у вигляді числових значень, індикаторів, графіків чи інших елементів, що полегшують сприйняття динаміки. Зворотний канал взаємодії реалізується через зміну певних полів у структурі даних Firebase, які інтерпретуються як налаштування або команди для вимірювального вузла. Таким чином, користувач, змінюючи параметри в мобільному додатку, фактично модифікує стан хмарної бази, а вже потім ці зміни враховуються у поведінці ESP32.

Розподіл функцій між компонентами системи є важливим аспектом її архітектурного проєктування. У запропонованій моделі найбільш ресурсоємні операції, пов’язані з довготривалим зберіганням даних, забезпеченням доступу множини клієнтів і підтриманням механізмів безпеки, зосереджені у Firebase. На ESP32 покладено задачі, що вимагають безпосередньої взаємодії з фізичним середовищем, швидкого реагування на його зміни та регулярного передавання інформації у хмару. Мобільний додаток, у свою чергу, відповідає за наближення складної структури системи до користувача, перетворюючи масиви числових даних у зручні для інтерпретації графіки, повідомлення й налаштування. Такий розподіл дозволяє уникнути надмірного навантаження на окремі компоненти, спрощує їхнє оновлення та технічний супровід, а також створює передумови для подальшого масштабування системи, наприклад шляхом додавання нових вимірювальних вузлів або альтернативних клієнтських застосунків без втручання у базову логіку інших елементів.

У підсумку архітектура системи «ESP32 – Firebase – Android» реалізує принцип побудови кластеризованих програмно-апаратних комплексів, у яких

кожний компонент є окремою прикладною програмою або сервісом, але разом вони утворюють єдине функціональне ціле. Саме така архітектурна модель буде покладена в основу подальшого проєктування структури даних у Firebase та розроблення програмної логіки окремих складових системи.

2.3 Проєктування структури даних у Firebase

Проєктування структури даних у хмарній базі Firebase є ключовим етапом побудови системи, оскільки саме від обраної моделі зберігання залежить зручність доступу до інформації, ефективність виконання запитів, простота розширення функціональності та масштабування. На відміну від реляційних баз даних, де структура визначається у вигляді таблиць з чітко заданими схемами, Firebase Realtime Database та Cloud Firestore використовують ієрархічну або документоорієнтовану модель зберігання, що потребує особливої уваги до організації гілок, колекцій та документів. У контексті системи «ESP32 – Firebase – Android» структура даних має відображати, з одного боку, вимірювання мікрокліматичних параметрів, а з іншого – інформацію про пристрої, користувачів та їхні налаштування.

Базовою сутністю в системі моніторингу мікроклімату є вимірювання, яке пов’язує конкретний вимірювальний вузол, часову мітку та значення параметрів середовища. Для зручності та узгодженості обробки доцільно представити кожне вимірювання у вигляді об’єкта, що містить поля для температури, відносної вологості, часу реєстрації та, за потреби, службових атрибутів (ідентифікатора пристрою, стану обробки тощо). Логічно згрупувати такі об’єкти можна за ознакою пристрою, на якому вони були сформовані. У найпростішому випадку для Realtime Database це може бути гілка верхнього рівня, наприклад devices, в якій для кожного пристрою зберігається власний підрозділ вимірювань. Всередині такого вузла записи можуть організовуватися за ключами, що відповідають часовим міткам у стандартному форматі, або за автоматично згенерованими ідентифікаторами, але з обов’язковим явним збереженням часу у полі об’єкта.

Окреме завдання становить моделювання інформації про самі пристрої. Оскільки система потенційно може працювати з кількома вимірювальними вузлами, доцільно створити окрему гілку або колекцію, у якій для кожного пристрою зберігатимуться його статичні характеристики: унікальний ідентифікатор, опис розташування (аудиторія, кімната, лабораторія), типи підключених сенсорів, а також поточний стан (активний, недоступний, тестовий режим тощо). Така структура дозволяє мобільному застосунку отримувати не лише числові значення вимірювань, а й контекст, у якому ці значення були сформовані, та за потреби відображати список доступних вузлів для користувача.

Важливим аспектом є представлення конфігураційних параметрів та керуючих команд. Оскільки Android-застосунок має надавати можливість змінювати налаштування системи – наприклад, порогові значення температури, при перевищенні яких слід формувати попередження, або період опитування сенсорів – ці параметри повинні бути доступні як для мобільного клієнта, так і для ESP32. Одним із підходів є виділення в структурі бази окремої гілки settings або config, у якій зберігаються глобальні налаштування системи, а також, за потреби, специфічні параметри для кожного окремого пристрою. ESP32 періодично зчитує поля, що відповідають його конфігурації, та коригує власну логіку роботи, тоді як мобільний застосунок модифікує ці поля через інтерфейс Firebase SDK.

Керуючі команди можуть бути представлені або як частина конфігураційних параметрів, або як окремий логічний канал обміну. У простому варіанті команда може бути закодована у вигляді зміни певного булевого або цілочислового поля, яке сигналізує про необхідність виконання конкретної дії, наприклад калібрування сенсорів чи примусового оновлення вимірювання. ESP32, зчитуючи це поле, має інтерпретувати його значення як тригер для відповідної процедури, після чого повертати параметр у початковий стан, фіксуючи виконання команди. Альтернативно можна використовувати список або чергу команд, де кожна команда є окремим об’єктом із власною часовою

міткою та типом, що дає змогу розширити набір доступних керуючих впливів без істотної зміни вже існуючої логіки.

Проєктуючи структуру даних, необхідно також враховувати специфіку доступу з боку мобільного застосунку. Застосунок Android має швидко отримувати поточні значення параметрів для відображення на головному екрані, а також мати можливість завантажити історію вимірювань за певний проміжок часу для побудови графіків. Це означає, що структура бази має підтримувати ефективні запити за часом та, можливо, обмеженням кількості повернутих записів. Для цього доцільно застосовувати часові мітки як ключі або поля, за якими можна виконувати впорядковані вибірки. У випадку використання Firestore як документоорієнтованої бази даних кожне вимірювання може бути окремим документом у колекції, впорядкованим за полем timestamp, що спрощує реалізацію запитів типу «останнє значення» або «усі значення за останню добу».

Окремий вимір проєктування структури даних пов’язаний із забезпеченням безпеки й розмежування доступу. Правила безпеки Firebase дозволяють задати, які користувачі або клієнти можуть читати чи змінювати певні гілки чи колекції. У найпростішому випадку система може передбачати один обліковий запис для адміністратора, який має повний доступ до всіх даних і налаштувань, тоді як інші користувачі обмежуються правами читання вимірювань і, можливо, коригування деяких локальних параметрів інтерфейсу. Для мікроконтролера ESP32, що діє як технічний клієнт, можна налаштувати окремий режим доступу, який дозволяє записувати вимірювання та читати конфігураційні поля, але не змінювати інші частини бази.

Узагальнюючи, структура даних у Firebase для системи «ESP32 – Firebase – Android» має забезпечувати логічне розділення вимірювань, інформації про пристрої, конфігураційних параметрів і керуючих команд, а також підтримувати ефективні сценарії доступу для мікроконтролера та мобільного клієнта. Гнучкість документоорієнтованої моделі дозволяє розширювати систему, додаючи нові типи даних або модулі, без радикальної перебудови існуючої схеми, що є важливою передумовою для подальшого розвитку й масштабування кластера прикладних програм.

2.4 Проєктування програмної логіки ESP32 та мобільного застосунку Android

Проєктування програмної логіки системи «ESP32 – Firebase – Android» передбачає узгоджене визначення алгоритмів роботи обох кінцевих компонентів кластера прикладних програм, а саме вимірювального вузла на мікроконтролері ESP32 та мобільного клієнта на платформі Android. Важливо, щоб обидві частини оперували єдиною моделлю даних, використовували погоджені формати повідомлень і синхронізували свої стани через хмарне середовище Firebase, не вимагаючи прямого мережевого з’єднання між собою. Тому вже на етапі проєктування логіки необхідно окреслити послідовність дій, які виконуються на кожному боці, а також точки, у яких відбувається їхня непряма взаємодія через базу даних.

На стороні ESP32 програмна логіка спирається на типову для вбудованих систем модель циклічного опитування та обробки подій. Після початкової ініціалізації апаратних ресурсів мікроконтролер переходить у основний цикл, у межах якого з певною періодичністю виконується опитування підключеного сенсора температури та відносної вологості, обробка отриманих значень та передавання сформованих даних у Firebase. На етапі проєктування вирішальним є вибір структури цього циклу, способу організації часових інтервалів та механізмів обробки помилок. Застосування примітивних затримок, що блокують виконання програми, може бути прийнятним лише в найпростіших сценаріях, тоді як більш гнучким рішенням є використання неблокувальних підходів, заснованих на відліку часу за допомогою системних таймерів. Це дозволяє, за потреби, додавати в прошивку додаткові функції без суттєвого впливу на стабільність основного циклу.

Важливою частиною логіки ESP32 є формування та надсилання запиту до хмарної бази даних. На рівні проєктування необхідно визначити, який саме формат повідомлення використовуватиметься, яким чином кодуватимуться температурні значення, відносна вологість і час, а також які допоміжні поля є

доцільними для подальшої обробки. Логіка має передбачати випадки недоступності мережі або хмарного сервісу, коли передавання даних не може бути здійснено. У таких ситуаціях доцільно передбачити повторні спроби встановлення з’єднання через певні інтервали або тимчасове буферизування обмеженої кількості вимірювань у оперативній пам’яті з подальшим їхнім надсиланням після відновлення зв’язку. Хоча повноцінна реалізація складних черг може виходити за межі базового прототипу, сама логічна структура прошивки має бути готовою до такого розширення.

Другим важливим аспектом програмної логіки ESP32 є обробка конфігураційних параметрів та керуючих команд, що зберігаються у Firebase. На етапі проєктування визначається, які саме параметри вузол має періодично зчитувати із хмари, як часто це має відбуватися та як зміна цих параметрів впливатиме на виконання основного циклу. Наприклад, якщо у Firebase задається період опитування сенсорів, що може змінюватися користувачем, мікроконтролер повинен враховувати нові значення без перезавантаження прошивки. Це передбачає виділення в логіці окремого блоку, відповідального за зчитування конфігурації, її валідацію та застосування. Керуючі команди можуть бути реалізовані як зміни певних полів, що розглядаються контролером як тригери для виконання разових операцій, після чого відповідні значення повертаються у «нейтральний» стан. Таким чином, логіка ESP32 включає два взаємодоповнювальні контури: контур передавання даних у хмару та контур зчитування налаштувань і команд з хмари.

Програмна логіка мобільного застосунку Android, своєю чергою, зосереджена на організації взаємодії користувача із системою через графічний інтерфейс і інтеграції з Firebase за допомогою офіційного програмного інтерфейсу. На етапі проєктування необхідно визначити основні екрани застосунку, їхню функціональну роль, способи відображення поточних значень і історії вимірювань, а також механізми внесення користувацьких змін у налаштування. Зазвичай передбачають хоча б головний екран з індикацією актуальних параметрів мікроклімату, окремий екран або вкладку для перегляду історії у вигляді графіків, а також форму для налаштування порогових значень

чи періоду опитування. На логічному рівні важливо, щоб усі ці елементи інтерфейсу зверталися до Firebase через узагальнений шар доступу до даних, а не напряму працювали з хмарною базою, що спрощує супровід і можливе розширення функціональності.

Асиметрія ролей між ESP32 та Android-застосунком знаходить відображення і в підході до роботи з Firebase. Якщо для мікроконтролера взаємодія з хмарою організована переважно як послідовність окремих запитів на запис або читання, ініційованих з певною періодичністю, то мобільний застосунок використовує реактивну модель, коли оновлення даних у інтерфейсі відбувається у відповідь на події, що надходять від Firebase. Це означає, що програмна логіка клієнта має містити окремі обробники подій зміни відповідних вузлів або документів, які відповідають за синхронізацію стану користувацького інтерфейсу з поточним станом хмарної бази. Проєктуючи таку логіку, важливо передбачити, які саме частини інтерфейсу повинні оновлюватися при зміні певних даних, як обробляти тимчасову недоступність мережі, а також як забезпечити коректне відновлення стану після повторного з’єднання.

Не менш важливим елементом логіки Android-застосунку є реалізація механізмів внесення змін у конфігураційні параметри системи. У більшості випадків ці зміни здійснюються через форми вводу або елементи керування, значення яких після підтвердження користувачем записуються у відповідні поля Firebase. На логічному рівні така операція розглядається як транзакція, що змінює стан хмарної бази і, відповідно, впливає на подальшу поведінку ESP32. На етапі проєктування потрібно визначити, які саме поля можуть змінюватися користувачем, які обмеження накладаються на їхні значення та як у інтерфейсі відображати факт успішного застосування налаштувань або повідомляти про можливі помилки чи відмову у записі.

Таким чином, програмна логіка ESP32 і мобільного застосунку Android проєктуються як дві взаємодоповнювальні частини єдиної системи, що взаємодіють між собою через проміжну хмарну ланку у вигляді Firebase. Мікроконтролер забезпечує перетворення фізичних величин на дані, придатні для зберігання та аналізу, тоді як Android-застосунок виконує роль інтерфейсу

користувача й засобу керування параметрами роботи системи. Узгодженість цих двох логік, а також їхня прив’язка до узагальненої моделі даних у Firebase, створюють основу для реалізації кластера прикладних програм, спроєктованого з урахуванням вимог до розподілених мобільно-IoT систем.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ РОБОТИ СИСТЕМИ

3.1 Підготовка інструментів розробки та налаштування середовищ

На першому етапі реалізації системи було підготовлено інструментальне середовище для всіх трьох компонентів розробки: мікроконтролера ESP32, хмарної платформи Firebase та мобільного застосунку Android. Це дозволило надалі зосередитися безпосередньо на програмній логіці, а не на технічних труднощах із налаштуванням.

Для програмування ESP32 було обрано середовище Arduino IDE, яке забезпечує достатній набір засобів для написання, компіляції та завантаження скетчів. Після встановлення Arduino IDE у меню «Файл → Налаштування» в поле «Додаткові URL менеджера плат» було додано адресу офіційного репозиторію плат ESP32. Далі через «Інструменти → Плата → Менеджер плат…» було знайдено пакет ESP32 і встановлено його. Після цього у меню вибору плати стала доступною низка конкретних моделей ESP32, серед яких було обрано ту, що відповідає використаному апаратному модулю.

Підключивши плату ESP32 до комп’ютера через USB-кабель, було встановлено драйвери послідовного порту, після чого в Arduino IDE вказано відповідний COM-порт у меню «Інструменти → Порт». Для первинної перевірки коректності налаштувань був створений мінімальний тестовий скетч, який виводив повідомлення у послідовний монітор. Це дозволило переконатися, що компіляція та завантаження програми відбуваються успішно, а плата коректно визначається системою (рис. 3.1).

Після завантаження цього скетчу та відкриття послідовного монітора були отримані очікувані повідомлення, що підтвердило коректність налаштування середовища розробки для ESP32.

Наступним кроком було створення хмарного проєкту у Firebase. У веб-консолі Firebase було створено новий проєкт, у межах якого активовано сервіс Firestore. Після ініціалізації бази даних і вибору регіону було згенеровано базову структуру правил доступу (на етапі розробки вони можуть бути спрощеними). У

консолі Firebase були зафіксовані основні параметри проєкту – projectId, URL бази даних та ключ API, які надалі використовуються на стороні Android-застосунку і, за потреби, у прошивці ESP32 для формування HTTP-запитів.

```
// Тестовий скетч для перевірки ESP32

void setup() {
  Serial.begin(115200);           // Ініціалізація послідовного порту
  Serial.println();
  Serial.println("ESP32 test is running...");
}

void loop() {
  Serial.println("Hello from ESP32!");
  delay(1000);                    // Раз на секунду виводимо повідомлення
}
```

Рисунок 3.1 – Тестовий скетч для виводу повідомлення у послідовний монітор

Паралельно було підготовлено середовище розробки мобільного застосунку в Android Studio. Було створено новий Android-проєкт із головною активністю, типу Empty Activity, для якого задано унікальний ідентифікатор пакета, що надалі використовується при інтеграції з Firebase. У консолі Firebase було додано Android-додаток до створеного проєкту, вказано той самий ідентифікатор пакета, після чого завантажено файл конфігурації google-services.json і додано його до структури Android-проєкту. У файли build.gradle рівня проєкту та модуля було внесено зміни, пов’язані з підключенням плагіна Google Services та залежностей Firebase, після чого проєкт було синхронізовано.

Після виконання цих кроків середовище було готове до подальшої реалізації логіки: ESP32 міг бути запрограмований у Arduino IDE, Firebase – приймати й зберігати дані, а Android-застосунок – взаємодіяти з хмарною базою через офіційний SDK. Подальші кроки були зосереджені на створенні прошивки вимірювального вузла на базі ESP32.

3.2 Програмна реалізація мікроконтролера ESP32

Після підготовки інструментів розробки було розпочато поетапну розробку прошивки для ESP32. Метою цього етапу стало створення програми, яка підключається до Wi-Fi-мережі, періодично зчитує показники з датчика температури та вологості, формує JSON-повідомлення і передає його до Firebase через HTTP(S).

Для цього в Arduino IDE було створено новий скетч (рис. 3.2).

```
#include <WiFi.h>           // Робота з Wi-Fi на ESP32
#include <HTTPClient.h>     // Формування HTTP-запитів
#include "DHT.h"            // Бібліотека для датчика DHT22

// Налаштування Wi-Fi
const char* WIFI_SSID     = "your-ssid";
const char* WIFI_PASSWORD = "your-password";

// Параметри DHT22
#define DHTPIN  4          // GPIO, до якого підключено DHT22
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// Параметри Firebase (REST API Realtime Database)
const char* FIREBASE_HOST = "https://your-project-id-default-rtdb.europe-west1.firebasedatabase.ap
const char* DEVICE_ID     = "device1";

// Змінні для таймінгу вимірювань
unsigned long previousMillis = 0;
const unsigned long MEASURE_INTERVAL = 10000; // 10 секунд
```

Рисунок 3.2 – Новий скетч з підключеними бібліотеками

На першому кроці я підключив необхідні бібліотеки: стандартну бібліотеку WiFi для ESP32, бібліотеку HTTPClient для формування HTTP-запитів, а також бібліотеку для роботи з датчиком DHT22. Після цього було оголошено базові константи: назву Wi-Fi мережі, пароль, тип датчика, номер GPIO-виводу, до якого він під’єднаний, URL хмарної бази Firebase та ідентифікатор пристрою. Додатково були введені змінні для реалізації неблокувального таймінгу вимірювань.

На цьому етапі особливу увагу було приділено саме оголошенню констант і структурі початкових налаштувань, оскільки вони безпосередньо впливають на зручність подальшої модифікації прошивки. Дані для підключення до Wi-Fi (SSID і пароль) винесені в окремі змінні, щоб за потреби змінювати їх без втручання в основну логіку коду. Аналогічно URL Firebase ідентифікує конкретний проєкт і регіон, тому його зручно мати у вигляді єдиної константи FIREBASE_HOST. Ідентифікатор пристрою DEVICE_ID дозволяє відрізняти записи від різних вузлів у базі даних і легко масштабувати систему на кілька ESP32.

Важливою деталлю є вибір інтервалу вимірювань. Константа MEASURE_INTERVAL задає періодичність зчитування сенсора у мілісекундах. Значення 10 секунд було обрано як компроміс: з одного боку, цього достатньо для відстеження змін мікроклімату в приміщенні, які відбуваються відносно повільно, а з іншого – не створюється надмірне навантаження на Wi-Fi-мережу та хмарну базу. Крім того, саме винесення інтервалу в окрему змінну дозволяє надалі керувати ним із Firebase, не змінюючи прошивку.

Після підготовки оголошень було реалізовано функцію ініціалізації setup(), яка виконується один раз при старті пристрою. На цьому кроці здійснюється налаштування послідовного порту для виведення діагностичних повідомлень, запуск бібліотеки DHT та встановлення Wi-Fi-з’єднання (рис. 3.3).

На практиці саме на цьому етапі кілька разів перевірено коректність введених параметрів мережі, оскільки будь-яка помилка в SSID чи паролі призводить до нескінченної спроби підключення. Послідовний монітор Arduino IDE став основним засобом налагодження: по послідовності крапок, що виводяться в циклі while, можна спостерігати процес підключення, а повідомлення з IP-адресою підтверджує успішне встановлення з’єднання. Також тестувався і сам датчик DHT22: у випадку некоректного підключення або відсутності ініціалізації функції зчитування пізніше повертали б значення NaN.

```
void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.println("Starting ESP32 IoT node...");

  // Ініціалізація датчика
  dht.begin();

  // Підключення до Wi-Fi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println();
  Serial.print("WiFi connected, IP address: ");
  Serial.println(WiFi.localIP());
}
```

Рисунок 3.3 – Ініціалізація послідовного порту, датчика DHT22 та встановлення з’єднання з Wi-Fi

Після того, як базова ініціалізація була виконана, основну логіку вимірювання та передавання даних було розміщено у функції loop() (рис. 3.4). Тут реалізовано неблокувальний підхід до таймінгу: замість використання довгих delay() контролер орієнтується на різницю між поточним часом (функція millis()) та часом останнього вимірювання. Це дозволяє за потреби додавати до прошивки інші задачі, не заважаючи регулярному опитуванню сенсора.

На цьому етапі у послідовному моніторі вже можна було спостерігати перші «живі» дані – температуру і вологість, що підтверджувало коректність роботи датчика DHT22, його підключення до GPIO і налаштування бібліотеки. Паралельно було перевірено поведінку у випадку помилок: наприклад, при відсутності живлення на датчику або при поганому контакті бібліотека повертає значення NaN, що фіксується у перевірці isnan(...), і програма коректно пропускає спробу надсилання некоректних даних.

```
void loop() {
  unsigned long currentMillis = millis();

  // Час виконати нове вимірювання
  if (currentMillis - previousMillis >= MEASURE_INTERVAL) {
    previousMillis = currentMillis;

    float temperature = dht.readTemperature();   // °C
    float humidity    = dht.readHumidity();      // %

    // Перевірка на коректність вимірювання
    if (isnan(temperature) || isnan(humidity)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

    Serial.print("T = ");
    Serial.print(temperature, 1);
    Serial.print(" °C, H = ");
    Serial.print(humidity, 1);
    Serial.println(" %");

    // Формування часової мітки (спрощено – millis)
    unsigned long timestamp = currentMillis;

    // Надсилання даних до Firebase
    sendToFirebase(temperature, humidity, timestamp);
  }
}
```

Рисунок 3.4 – Основна логіка вимірювання та передавання даних у функції loop()

Ключовою частиною прошивки є функція sendToFirebase, яка реалізує безпосередню взаємодію з хмарною базою Firebase через REST-інтерфейс Realtime Database. У процесі реалізації було обрано схему, за якою кожне нове вимірювання записується як окремий об’єкт у гілку /devices/{deviceId}/measurements. Це дозволяє зберігати історію вимірювань і надалі будувати графіки, статистику тощо.

На етапі первинного тестування каналу передавання даних доцільно було ізолювати логіку взаємодії з Firebase від реального сенсора й перевірити, чи коректно працює саме зв’язка «ESP32 – Wi-Fi – Firebase». Для цього було створено тестовий фрагмент коду, у якому замість фактичних показників DHT22 використовуються фіксовані або умовні значення температури та вологості. Один раз після старту контролера формується JSON-повідомлення, що імітує реальне вимірювання, і надсилається у базу даних. Такий підхід дав змогу окремо перевірити правильність URL, структуру JSON, заголовки HTTP-запиту та реакцію сервера, не враховуючи можливі збої сенсора.

У фрагменті коду в лістингу 3.1 наведено спрощену тестову версію, яка демонструє формування пакета даних і його відправлення до Firebase одразу після підключення до Wi-Fi. У послідовному моніторі Arduino IDE при цьому відображаються як відлагоджувальні повідомлення з параметрами вимірювання, так і код відповіді сервера, що дозволяє оперативно переконатися у коректності конфігурації.

Лістинг 3.1 – Формування пакета даних та відправка до Firebase

```
#include <WiFi.h>
#include <HTTPClient.h>

// Ті самі параметри Wi-Fi та Firebase, що й у основній прошивці
const char* WIFI_SSID     = "your-ssid";
const char* WIFI_PASSWORD = "your-password";
const char* FIREBASE_HOST = "https://your-project-id-default-rtdb.europe-west1.firebasedatabase.app";
const char* DEVICE_ID     = "device1";

// Формування JSON-повідомлення для тестового вимірювання
String buildMeasurementJson(float temperature, float humidity, unsigned long timestamp) {
  String payload = "{";
  payload += "\"temperature\":" + String(temperature, 1) + ",";
  payload += "\"humidity\":"    + String(humidity, 1)    + ",";
  payload += "\"timestamp\":"   + String(timestamp);
  payload += "}";
  return payload;
}

// Надсилання готового JSON-повідомлення до Firebase
void sendToFirebase(const String& payload) {
```

```
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("WiFi not connected, skipping send.");
    return;
  }

  HTTPClient http;
  String url = String(FIREBASE_HOST) +
               "/devices/" + DEVICE_ID +
               "/measurements.json";

  http.begin(url);
  http.addHeader("Content-Type", "application/json");

  Serial.print("POST ");
  Serial.println(url);
  Serial.print("Payload: ");
  Serial.println(payload);

  int httpCode = http.POST(payload);

  if (httpCode > 0) {
    Serial.print("Firebase response code: ");
    Serial.println(httpCode);
    String response = http.getString();
    Serial.println("Response: " + response);
  } else {
    Serial.print("Error on sending POST: ");
    Serial.println(http.errorToString(httpCode));
  }

  http.end();
}

void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.println("Test send to Firebase...");

  // Підключення до Wi-Fi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("WiFi connected, IP: ");
  Serial.println(WiFi.localIP());

  // Умовні (тестові) значення температури та вологості
  float testTemperature = 25.0;
  float testHumidity    = 40.0;
  unsigned long timestamp = millis();
```

```
  Serial.print("Test T = ");
  Serial.print(testTemperature, 1);
  Serial.print(" °C, H = ");
  Serial.print(testHumidity, 1);
  Serial.println(" %");

  // Формуємо та надсилаємо тестовий JSON
  String payload = buildMeasurementJson(testTemperature, testHumidity, timestamp);
  sendToFirebase(payload);
}

void loop() {
  // У тестовому прикладі основний цикл порожній
}
```

Кінець лістингу 3.1

Після завантаження цього тестового скетчу на ESP32 у послідовному моніторі фіксується процес підключення до бездротової мережі, виводяться умовні значення температури та вологості, сформований JSON-пакет і код відповіді сервера Firebase. В інтерфейсі Realtime Database (вузол /devices/device1/measurements) з’являється новий об’єкт із зазначеними полями, що підтверджує працездатність каналу передавання даних. Лише після успішного проходження цього тесту до коду було повернено інтеграцію з реальним датчиком DHT22 та розширено логіку циклічних вимірювань.

3.3 Реалізація структури даних та правил доступу у Firebase

Після того як мікроконтролер ESP32 навчився передавати вимірювання у Firebase, наступним логічним кроком стало впорядкування цих даних у хмарі та налаштування правил доступу. На цьому етапі можна спостерігати у Realtime Database гілку з автоматично згенерованими ключами, але структура була «сира» і не до кінця відображала модель, спроєктовану в підрозділі 2.3. Тому завданням стало привести базу до логічно впорядкованого вигляду й одночасно задати мінімально необхідні правила безпеки.

У веб-консолі Firebase в розділі Realtime Database було створено кореневу гілку devices, у якій зберігається інформація про всі вузли системи. Після

першого запуску тестового та основного скетчів ESP32 у гілці devices автоматично з’явився вузол з ідентифікатором device1, а всередині нього – підгілка measurements із вимірюваннями, які надсилала функція sendToFirebase. Фрагмент реальної структури даних на цьому етапі показано на рисунку 3.5.


```
{
  "devices": {
    "device1": {
      "measurements": {
        "-Nabc123xyz": {
          "temperature": 24.7,
          "humidity": 39.8,
          "timestamp": 305000
        },
        "-Nabc456uvw": {
          "temperature": 24.6,
          "humidity": 40.1,
          "timestamp": 315000
        }
      }
    }
  }
}
```


Рисунок 3.5 – Фрагмент структури даних

На основі цього «живого» дерева даних у консолі я додатково створив ще дві логічні підгілки: config для зберігання конфігураційних параметрів кожного пристрою та, за потреби, meta для зберігання описової інформації (розташування, тип сенсорів тощо). Їх було спочатку додано вручну через вкладку Data, щоб мати чіткий орієнтир для подальшого коду і мобільного застосунку. Структура наблизилася до такого вигляду (рис. 3.6):

Після того як базові гілки були створені, я перевірив, як це вплине на роботу існуючої прошивки. Оскільки ESP32 надсилає дані виключно в devices/device1/measurements, додавання meta і config не вимагало змін у коді. Навпаки, з’явилася можливість у наступних версіях прошивки реалізувати читання параметра measureIntervalMs із хмари замість жорстко заданої константи MEASURE_INTERVAL у скетчі. Для цього в консолі навмисне кілька

разів змінював значення measureIntervalMs, щоб переконатися, що Android-застосунок надалі зможе відобразити поточні налаштування, а ESP32 – підлаштувати період опитування.


```
{
  "devices": {
    "device1": {
      "meta": {
        "location": "Лабораторія 101",
        "description": "Тестовий вузол моніторингу мікроклімату",
        "sensorType": "DHT22"
      },
      "config": {
        "measureIntervalMs": 10000,
        "tempWarningThreshold": 28.0,
        "humidityWarningThreshold": 70.0
      },
      "measurements": {
        "-Nabc123xyz": {
          "temperature": 24.7,
          "humidity": 39.8,
          "timestamp": 305000
        }
      }
    }
  }
}
```


Рисунок 3.6 – Оновлена структура

Наступним кроком стало налаштування правил безпеки Firebase. На етапі найпершого тестування база могла працювати з максимально спрощеними правилами («.read»: true, «.write»: true), однак для реальної експлуатації навіть у навчальній лабораторії це є небажаним. Тому перейшов у вкладку Rules і поступово звузив можливості анонімного доступу. На початковому етапі було задано мінімальне обмеження: дозволити читання для всіх, але запис – лише для автентифікованих клієнтів. Приклад правил мав вигляд (рис. 3.7):

```
{
  "rules": {
    ".read": true,
    ".write": "auth != null"
  }
}
```

Рисунок 3.7 – Налаштування правил безпеки Firebase

Після застосування цих правил я свідомо перевірив роботу системи: Android-застосунок, який планувалося інтегрувати з Firebase Authentication, зберігав можливість читати дані, а спроби запису з неавторизованих клієнтів (наприклад, через сторонні інструменти REST) блокувалися. Водночас ESP32, який у базовій конфігурації використовує відкритий доступ за прямим URL, міг зіткнутися з обмеженнями. Тому для вузлів, які заповнюються саме мікроконтролером, було налаштовано окремі правила з прив'язкою до конкретної гілки.

На наступному етапі деталізував правила для гілки devices. Ідея полягала в тому, щоб дозволити запис у measurements лише з боку «сервісного» клієнта (яким у перспективі може бути або ESP32 із використанням секретного ключа/токена, або проміжний бекенд), а мобільним користувачам – тільки читання вимірювань та обмежений запис у config. Приклад більш деталізованих правил для Realtime Database виглядав так (рис. 3.8):

Хоча у рамках базового прототипу рольова модель (auth.token.role) може не бути реалізованою повністю, саме таке формулювання правил відображає цільову архітектуру доступу, у якій різні типи клієнтів мають різний рівень прав. На практиці, тестуючи ці правила, спочатку спростив умови до варіантів auth != null, а вже потім поступово ускладнював їх, щоб переконатися, що ні мобільний застосунок, ні ESP32 не втрачають необхідний доступ.

На завершення налаштування структури даних і правил безпеки провів серію перевірок через веб-консоль Firebase:

– спостерігав, як ESP32 додає нові об'єкти у measurements;

– вручну змінював параметри у вузлі config (наприклад, measureIntervalMs), готуючи базу до подальшої інтеграції з Android-застосунком;

– тестував поведінку при зміні правил (повне закриття запису, обмеження читання тощо), відстежуючи відповідні помилки в логах.

```
{
  "rules": {
    "devices": {
      "$deviceId": {
        "measurements": {
          ".read": true,
          ".write": "auth != null && auth.token.role == 'device'"
        },
        "config": {
          ".read": "auth != null",
          ".write": "auth != null && auth.token.role == 'admin'"
        },
        "meta": {
          ".read": true,
          ".write": "auth != null && auth.token.role == 'admin'"
        }
      }
    }
  }
}
```

Рисунок 3.8 – Правила для Realtime Database

У результаті структура даних у Firebase набула цілісного вигляду, що напряму відповідає логічній моделі, описаній у другому розділі, а правила доступу сформували каркас безпеки, достатній для експериментальної експлуатації системи. Це стало основою для наступного кроку – реалізації мобільного застосунку Android, який має використовувати цю структуру даних для візуалізації вимірювань та керування налаштуваннями системи.

3.4 Реалізація мобільного застосунку Android для роботи з Firebase

Після того як структура даних у Firebase була впорядкована, а вимірювальний вузол на ESP32 почав стабільно надсилати вимірювання в гілку /devices/device1/measurements, наступним кроком стала реалізація мобільного застосунку Android. На цьому етапі я вже міг спостерігати «живі» дані у консолі

Firebase, тому завданням Android-клієнта було навчитися підключатися до тієї ж бази, зчитувати актуальні значення та відображати їх користувачеві в зручній формі.

Розробку почав із відкриття підготовленого раніше проєкту в Android Studio, в якому вже було додано файл google-services.json та підключено базову інтеграцію з Firebase. Першим практичним кроком стало додавання залежностей для роботи з Realtime Database у файл build.gradle модуля додатка. Для уникнення проблем із версіями було використано Firebase BoM (рис. 3.9).

Після синхронізації проєкту Android Studio підтвердила, що бібліотека firebase-database-ktx доступна, а отже, можна переходити до безпосередньої взаємодії з Firebase із коду застосунку. Додаткової ініціалізації у вигляді виклику FirebaseApp.initializeApp(...) у більшості випадків не потрібно, оскільки плагін google-services та конфігураційний файл google-services.json роблять це автоматично при старті додатка.

Далі перейшов до проєктування найпростішого інтерфейсу для відображення поточних значень температури та відносної вологості. У файлі розмітки activity_main.xml були додані два текстові поля для показу температури й вологості та допоміжний текст для індикатора стану підключення/завантаження даних (лістинг 3.2).

```
// build.gradle (Module: app)

dependencies {
    // Платформа Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:33.1.0')

    // Бібліотека Realtime Database
    implementation 'com.google.firebase:firebase-database-ktx'

    // Інші стандартні залежності Android
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.11.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
}
```

Рисунок 3.9 – Реалізація мобільного застосунку Android

Лістинг 3.2 – Інтерфейс для відображення поточних значень температури та відносної вологості

```
<!-- res/layout/activity_main.xml -->
<androidx.constraintlayout.widget.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="16dp">

  <TextView
    android:id="@+id/tvTemperature"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="T: --.- °C"
    android:textSize="24sp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"/>

  <TextView
    android:id="@+id/tvHumidity"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="H: --.- %"
    android:textSize="24sp"
    app:layout_constraintTop_toBottomOf="@id/tvTemperature"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="16dp"/>

  <TextView
    android:id="@+id/tvStatus"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Завантаження даних..."
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/tvHumidity"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="16dp"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Кінець лістингу 3.2

Після цього перейшов до головної активності MainActivity, де реалізував логіку підключення до Firebase та зчитування даних. Спочатку було створено

модель вимірювання у вигляді data class, щоб зручно відображати об’єкти Realtime Database у Kotlin-коді (рис. 3.10).

```
// Простий клас-модель для вимірювання
data class Measurement(
    val temperature: Double? = null,
    val humidity: Double? = null,
    val timestamp: Long? = null
)
```

Рисунок 3.10 – Реалізація логіки підключення до Firebase

Потім, у MainActivity, отримав посилання на вузол /devices/device1/measurements і організував читання останнього за часом вимірювання за допомогою запиту orderByChild("timestamp").limitToLast(1). Весь код логіки для зчитування та відображення показано у лістингу 3.3.

Лістинг 3.3 – Код логіки зчитування та відображення

```
// MainActivity.kt
class MainActivity : AppCompatActivity() {

    private lateinit var tvTemperature: TextView
    private lateinit var tvHumidity: TextView
    private lateinit var tvStatus: TextView

    // Посилання на Firebase Realtime Database
    private val database = FirebaseDatabase.getInstance()
    private val measurementsRef = database
        .getReference("devices")
        .child("device1")
        .child("measurements")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        tvTemperature = findViewById(R.id.tvTemperature)
        tvHumidity    = findViewById(R.id.tvHumidity)
        tvStatus      = findViewById(R.id.tvStatus)

        tvStatus.text = "Завантаження даних..."
```

```
        subscribeToLastMeasurement()
    }

    private fun subscribeToLastMeasurement() {
        // Отримуємо останнє за часом вимірювання
        val query = measurementsRef
            .orderByChild("timestamp")
            .limitToLast(1)

        query.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                if (!snapshot.hasChildren()) {
                    tvStatus.text = "Дані відсутні"
                    return
                }

                // У гілці може бути кілька дітей, беремо перший
                for (child in snapshot.children) {
                    val m = child.getValue(Measurement::class.java)
                    if (m != null && m.temperature != null && m.humidity != null) {
                        tvTemperature.text = "T: %.1f °C".format(m.temperature)
                        tvHumidity.text    = "H: %.1f %%".format(m.humidity)
                        tvStatus.text      = "Оновлено: ${m.timestamp}"
                    } else {
                        tvStatus.text = "Помилка читання даних"
                    }
                    break
                }
            }

            override fun onCancelled(error: DatabaseError) {
                tvStatus.text = "Помилка: ${error.message}"
            }
        })
    }
}
```

Кінець лістингу 3.3

Після додавання цього коду я зібрав і встановив застосунок на тестовий Android-пристрій, який був підключений до тієї ж мережі Інтернет, що й ESP32, але без прямої взаємодії між ними – усе відбувалося через Firebase. У процесі тестування спочатку відкривався екран із повідомленням «Завантаження даних…», а через кілька секунд, після обробки події onDataChange, інтерфейс оновлювався: у полях T: та H: з’являлися актуальні значення температури та відносної вологості, які були щойно надіслані мікроконтролером.

Цей крок був важливим з точки зору перевірки всієї зв’язки «ESP32 – Firebase – Android». Я паралельно відкривав Realtime Database у браузері та спостерігав, як у гілці measurements з’являються нові записи; майже одночасно з цим мобільний застосунок оновлював показники на екрані. Таким чином, було підтверджено, що:

– ESP32 коректно формує JSON-повідомлення і додає нові вимірювання до бази;

– структура даних у Firebase відповідає моделі Measurement;

Android-застосунок успішно підписується на зміни й може реагувати на появу нових записів у режимі, наближеному до реального часу.

На цьому етапі мобільний клієнт реалізовував базову функцію – відображення останнього вимірювання. У подальших кроках (за потреби розширення системи) до застосунку можна додати:

– екран для побудови графіка історії вимірювань за певний інтервал часу;

– інтерфейс для редагування конфігураційних параметрів у гілці config (наприклад, порогів температури);

– механізми візуального попередження при виході параметрів за допустимі межі.

Однак уже на поточному рівні реалізації було досягнуто основної мети: мобільний застосунок Android став повноцінним елементом кластера прикладних програм, що працює з тими ж даними, які генерує ESP32 й зберігає Firebase.

Додатково в межах реалізації мобільного застосунку було створено окремий екран для візуалізації історії вимірювань у вигляді графіка. Такий екран дозволяє не лише бачити актуальні значення температури та вологості, а й аналізувати динаміку їх змін за останній проміжок часу. Для побудови лінійних графіків було використано бібліотеку MPAndroidChart, яка добре інтегрується з типовим стеком Android + Firebase.

Спочатку до файлу build.gradle модуля додатка було додано залежність на MPAndroidChart (рис. 3.11):

```
dependencies {
    implementation platform('com.google.firebase:firebase-bom:33.1.0')
    implementation 'com.google.firebase:firebase-database-ktx'

    // Бібліотека для побудови графіків
    implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'

    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.11.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
}
```

Рисунок 3.11 – Файл build.gradle модуля додатка

Після синхронізації проєкту було створено окремий екран з графіком у файлі розмітки activity_chart.xml. На ньому розміщено компонент LineChart, який відображатиме зміну температури та вологості в часі (рис. 3.12):

```
<!-- res/layout/activity_chart.xml -->
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <com.github.mikephil.charting.charts.LineChart
        android:id="@+id/chartMeasurements"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 3.12 – Екран з графіком у файлі розмітки activity_chart.xml

Далі було реалізовано окрему активність ChartActivity, яка зчитує з Firebase кілька останніх вимірювань (наприклад, 50 записів), перетворює їх у точки на графіку та будує дві лінії – для температури та вологості. У коді нижче використовується та сама модель Measurement, що й на головному екрані (лістинг 3.4):

Лістинг 3.4 – Код побудови графіків для температури та вологості

```
// ChartActivity.kt
class ChartActivity : AppCompatActivity() {

    private lateinit var chart: LineChart

    private val database = FirebaseDatabase.getInstance()
    private val measurementsRef = database
        .getReference("devices")
        .child("device1")
        .child("measurements")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_chart)

        chart = findViewById(R.id.chartMeasurements)

        setupChart()
        loadLastMeasurements()
    }

    private fun setupChart() {
        chart.description.isEnabled = false
        chart.setTouchEnabled(true)
        chart.setPinchZoom(true)
        chart.axisRight.isEnabled = false
    }

    private fun loadLastMeasurements() {
        // Отримуємо, наприклад, останні 50 вимірювань
        val query = measurementsRef
            .orderByChild("timestamp")
            .limitToLast(50)

        query.addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                val tempEntries = mutableListOf<Entry>()
                val humEntries  = mutableListOf<Entry>()

                // Щоб осі були «гарні», нормалізуємо час відносно мінімального
timestamp
```

```
                var minTimestamp: Long? = null

                val measurements = snapshot.children.mapNotNull { it.getValue(Measurement::class.java) }
                    .sortedBy { it.timestamp }

                if (measurements.isEmpty()) {
                    Toast.makeText(this@ChartActivity, "Немає даних для побудови графіка", Toast.LENGTH_SHORT).show()
                    return
                }

                minTimestamp = measurements.first().timestamp

                measurements.forEach { m ->
                    val t = m.timestamp ?: return@forEach
                    val x = (t - minTimestamp).toFloat() / 1000f // секунди від початку

                    val temp = m.temperature?.toFloat()
                    val hum  = m.humidity?.toFloat()

                    if (temp != null) {
                        tempEntries.add(Entry(x, temp))
                    }
                    if (hum != null) {
                        humEntries.add(Entry(x, hum))
                    }
                }

                showChart(tempEntries, humEntries)
            }

            override fun onCancelled(error: DatabaseError) {
                Toast.makeText(this@ChartActivity, "Помилка: ${error.message}", Toast.LENGTH_SHORT).show()
            }
        })
    }

    private fun showChart(
        tempEntries: List<Entry>,
        humEntries: List<Entry>
    ) {
        val tempDataSet = LineDataSet(tempEntries, "Температура (°C)")
        tempDataSet.lineWidth = 2f
        tempDataSet.setDrawCircles(false)

        val humDataSet = LineDataSet(humEntries, "Вологість (%)")
        humDataSet.lineWidth = 2f
        humDataSet.setDrawCircles(false)

        // За замовчуванням у MPAndroidChart усі лінії одного кольору,
        // але стилізацію можна налаштовувати окремо, якщо потрібно.
```

```
            val lineData = LineData(tempDataSet, humDataSet)
            chart.data = lineData
            chart.invalidate()
        }
    }
```

Кінець лістингу 3.4

На практиці перехід до цього екрану може виконуватися, наприклад, з головної активності через кнопку «Графік» або пункт меню. У найпростішому випадку до MainActivity достатньо додати обробник натискання, який відкриває ChartActivity за допомогою стандартного startActivity(Intent(this, ChartActivity::class.java)).

Після додавання екрану з графіком було проведено тестування: при наявності накопичених вимірювань у вузлі measurements користувач може перейти на екран історії та побачити два графіки, які відображають зміну температури та відносної вологості в часі. Це не змінює основної логіки роботи системи, але суттєво розширює її аналітичні можливості, наближаючи розроблений мобільний застосунок до повноцінного інструмента моніторингу мікроклімату.

3.5 Інтеграційне тестування та експериментальні дослідження роботи системи

Після окремого налагодження прошивки ESP32, структури даних у Firebase та мобільного застосунку Android було проведено інтеграційне тестування всієї системи в цілому. На цьому етапі метою було не лише підтвердити працездатність кожного з компонентів, а й оцінити, як вони поводяться в реальних умовах спільної роботи: чи узгоджуються часові характеристики, чи немає втрати даних, наскільки швидко мобільний застосунок реагує на нові вимірювання, а також чи коректно відображається інформація для користувача.

Спочатку було організовано базовий експеримент у лабораторному приміщенні. Вузол на базі ESP32 з підключеним датчиком DHT22 було розміщено на робочому столі, підключено до стабільного джерела живлення та приєднано до Wi-Fi-мережі, налаштованої на етапі розробки. В іншому кінці приміщення було розташовано Android-пристрій із встановленим мобільним застосунком, який підключався до Інтернету через ту ж мережу. У консолі Firebase в режимі реального часу було відкрито гілку devices/device1/measurements, що дозволяло спостерігати появу нових записів паралельно з відображенням даних у мобільному інтерфейсі.

Для початку було перевірено базову узгодженість даних між усіма трьома компонентами. Після запуску ESP32 контролювався процес підключення до Wi-Fi у послідовному моніторі Arduino IDE: виводилися службові повідомлення, IP-адреса пристрою та виміряні значення температури й вологості. Через кожні 10 секунд (відповідно до заданого інтервалу MEASURE_INTERVAL) у Firebase з’являвся новий об’єкт із полями temperature, humidity та timestamp. Мобільний застосунок, підписаний на останнє вимірювання за допомогою запиту orderByChild("timestamp").limitToLast(1), практично відразу після запису оновлював значення на екрані. Візуальне порівняння показників у послідовному моніторі, веб-консолі Firebase та інтерфейсі Android-застосунку дало змогу переконатися, що передавання та відображення даних відбувається без спотворень.

Далі було проведено спробу оцінити затримку між моментом вимірювання на ESP32 та відображенням оновлених значень у мобільному застосунку. Для цього у послідовному моніторі фіксувалися часові мітки (у мілісекундах від старту) при кожному формуванні вимірювання, а на смартфоні виводився timestamp, прочитаний із Firebase. Хоча така оцінка є наближеною, вона дозволила встановити, що при стабільному Wi-Fi-з’єднанні сумарна затримка (формування запиту, обробка на сервері, доставка змін клієнтові) зазвичай становила менше однієї секунди, що є цілком прийнятним для задач моніторингу мікроклімату.

Окрему увагу було приділено поведінці системи в умовах нестабільного мережевого з’єднання. Для цього проводився експеримент, під час якого Wi-Fi-точка доступу тимчасово вимикалася або пристрій ESP32 переносився в зону зі слабким сигналом. У логах послідовного монітора фіксувалося, що при втраті з’єднання спроби надсилання даних у Firebase завершувалися помилкою, про що свідчили відповідні повідомлення функції sendToFirebase. Після відновлення доступу до мережі контролер продовжував роботу, і нові вимірювання знову з’являлися в хмарі. При цьому базова реалізація не передбачала збереження «пропущених» вимірювань, але інтеграційне тестування показало, що логіка роботи достатньо стійка до тимчасових розривів: система не зависала й не вимагала ручного перезапуску.

Під час експериментів також перевірялася коректність відображення ситуацій, коли дані тимчасово відсутні. Для цього в одній із сесій тестування база measurements була очищена вручну через веб-консоль, після чого запускався мобільний застосунок. Логіка MainActivity передбачала перевірку на наявність даних у DataSnapshot; у випадку порожньої гілки в полі стану tvStatus відображалося повідомлення «Дані відсутні». Після запуску ESP32 нові вимірювання починали записуватися у Firebase, і додаток коректно реагував на появу першого запису, автоматично оновлюючи інтерфейс без необхідності перезапуску.

Ще один аспект інтеграційного тестування стосувався зміни конфігураційних параметрів у гілці config. Хоча повноцінне динамічне керування інтервалом опитування сенсора з боку Android-застосунку є темою для подальшого розширення системи, навіть на рівні експерименту було продемонстровано принципову можливість такого підходу. Значення measureIntervalMs у вузлі devices/device1/config змінювалося вручну через інтерфейс Firebase; у модифікованій версії прошивки ESP32 передбачалося періодичне зчитування цього параметра та оновлення внутрішньої змінної, що визначає інтервал вимірювання. Після цього в логах було видно, як період між послідовними вимірюваннями змінюється відповідно до нового значення в

конфігурації, що підтверджує працездатність зворотного каналу впливу через хмару.

У підсумку інтеграційні випробування показали, що розроблена система в обраному стеку «ESP32 – Firebase – Android» забезпечує стабільний цикл «вимірювання – передавання – зберігання – відображення» для параметрів мікроклімату приміщення. Усі три компоненти – апаратний вимірювальний вузол, хмарна база даних та мобільний клієнт – функціонують узгоджено, а структура даних у Firebase дозволяє як аналізувати історію вимірювань, так і готувати підґрунтя для подальшого розширення можливостей, таких як побудова графіків, реалізація сповіщень чи гнучкіше дистанційне керування параметрами системи.

ВИСНОВКИ

У першому розділі було розглянуто теоретичні засади побудови мобільно-орієнтованих систем Інтернету речей та проаналізовано роль окремих компонентів у загальній архітектурі кластера прикладних програм. Показано, що Інтернет речей формує багаторівневе середовище, у якому фізичні пристрої, мережна інфраструктура, хмарні платформи та клієнтські застосунки взаємодіють як елементи єдиної програмно-апаратної системи. У цьому контексті зв'язка «ESP32 – Firebase – Android» виступає конкретним прикладом реалізації такої архітектури, де мікроконтролер забезпечує збирання та первинну обробку даних, хмарна платформа виконує функції зберігання й синхронізації, а мобільний застосунок відповідає за візуалізацію та керування.

Було обґрунтовано доцільність використання мікроконтролера ESP32 як базової платформи для реалізації вузла Інтернету речей завдяки поєднанню вбудованого бездротового зв'язку, достатніх обчислювальних ресурсів та підтримки широкого спектра периферійних інтерфейсів. Розглянуто можливості Firebase як хмарної платформи типу Backend-as-a-Service, що надає інструменти для організації зберігання даних у режимі, наближеному до реального часу, реалізації механізмів безпеки та інтеграції з мобільними клієнтами. Окремо проаналізовано мобільну платформу Android як домінуючий клієнтський інтерфейс для IoT-рішень, здатний забезпечити користувачеві зручний доступ до даних і функцій системи та інкапсулювати складність взаємодії з хмарною інфраструктурою. Сукупність цих компонентів формує теоретичне підґрунтя для подальшого проєктування й реалізації кластера прикладних програм, орієнтованого на моніторинг параметрів середовища у вибраній предметній області.

У другому розділі було здійснено проєктування системи моніторингу параметрів мікроклімату приміщення на основі зв'язки «ESP32 – Firebase – Android» як прикладу кластера прикладних програм у мобільно-IoT середовищі. Окреслено предметну область та сформульовано постановку задачі, що передбачає забезпечення безперервного вимірювання температури й відносної

вологості, передавання даних до хмарної бази, їх збереження та візуалізацію у мобільному застосунку з можливістю впливу користувача на налаштування системи. Запропоновано трирівневу архітектуру, у якій ESP32 реалізує вимірювальний вузол, Firebase виконує функції хмарного сховища й посередника, а Android-застосунок виступає клієнтським інтерфейсом.

Розроблено логічну структуру даних у Firebase, що відображає вимірювання, інформацію про пристрої, конфігураційні параметри та керуючі команди, з урахуванням вимог до ефективного доступу з боку як мікроконтролера, так і мобільного клієнта. На основі цієї структури спроєктовано програмну логіку ESP32, яка поєднує циклічне опитування сенсорів, формування та надсилання вимірювань і обробку параметрів, що надходять із хмари, а також логіку Android-застосунку, орієнтовану на реактивну взаємодію з Firebase, візуалізацію даних та модифікацію налаштувань. Отримані результати створюють необхідний фундамент для наступного етапу роботи, присвяченого безпосередній реалізації розробленої архітектури, налаштуванню середовищ розробки та проведенню експериментального дослідження характеристик створеної системи.

У третьому розділі було послідовно реалізовано та випробувано кластер прикладних програм на основі зв’язки «ESP32 – Firebase – Android» для задачі моніторингу мікроклімату приміщення. На етапі підготовки було налаштовано середовище розробки: Arduino IDE для програмування ESP32, хмарний проєкт у Firebase та Android Studio для створення мобільного застосунку. Далі було розроблено прошивку мікроконтролера, яка забезпечує періодичне зчитування показників температури та відносної вологості, формування JSON-повідомлень і їх надсилання до Realtime Database через REST-інтерфейс.

У хмарній частині проєкту було сформовано логічну структуру даних з виділенням гілок для вимірювань, конфігураційних параметрів та метаданих пристрою, а також налаштовано базові правила безпеки, що розмежовують доступ до різних типів даних. На стороні мобільного клієнта було реалізовано Android-застосунок, який підключається до Firebase, зчитує останні вимірювання та відображає їх користувачеві в режимі, наближеному до реального часу.

Інтеграційні експерименти підтвердили працездатність усієї системи в цілому, показавши узгодженість показників між ESP32, Firebase і Android-застосунком, а також стійкість розроблених алгоритмів до тимчасових порушень мережевого з’єднання. Отримані результати демонструють, що обраний стек технологій дозволяє відносно просто реалізувати розподілену IoT-систему моніторингу на рівні прототипного проєкту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Biswal A., Subhashini J., Pasayat A. K. Air quality monitoring system for indoor environments using IoT // *AIP Conference Proceedings*. 2019. Vol. 2112, № 1. P. 020180. DOI: 10.1063/1.5112365 (дата звернення: 10.11.2022).

2. Beya F. Getting started with ESP32 and Firebase // *Firebase Developers – Medium*. 02.05.2021. URL: https://medium.com/firebase-developers/getting-started-with-esp32-and-firebase-1e7f19f63401 (дата звернення: 12.12.2022).

3. Pieters O., Deprost E., Van Der Donckt J. MIRRA: A Modular and Cost-Effective Microclimate Monitoring System for Real-Time Remote Applications // *Sensors*. 2021. Vol. 21, № 13. Article 4615. DOI: 10.3390/s21134615 (дата звернення: 18.01.2023).

4. Oo Z. L. et al. IoT-Based Weather Monitoring System Using Firebase Real Time Database with Mobile Application // *International Symposium on Environmental-Life Science and Nanoscales Technology*. Yangon, 27-28 December 2019. P. 1-6. URL: https://www.scirp.org/reference/referencespapers.aspx?referenceid=3411290 (дата звернення: 06.02.2023).

5. Espressif Systems. ESP32 Technical Reference Manual. URL: https://www.espressif.com/en/products/socs/esp32 (дата звернення: 12.02.2023).

6. Firebase. Firebase Realtime Database Documentation. URL: https://firebase.google.com/docs/database (дата звернення: 24.02.2023).

7. Firebase. Read and Write Data on Android (Realtime Database). URL: https://firebase.google.com/docs/database/android/read-and-write (дата звернення: 14.03.2023).

8. Android Developers. Guide to App Architecture. URL: https://developer.android.com/topic/architecture (дата звернення: 21.03.2023).

9. Santos R. ESP32: Firebase (Realtime Database) – Complete Guide // *Random Nerd Tutorials*. URL: https://randomnerdtutorials.com/esp32-firebase-realtime-database (дата звернення: 02.04.2023).

10. Philipp Jahoda. MPAndroidChart: A powerful & easy to use chart library for Android // GitHub. URL: https://github.com/PhilJay/MPAndroidChart (дата звернення: 14.04.2023).