Галицький фаховий коледж імені В'ячеслава Чорновола
Відділення комп'ютерних технологій
Циклова комісія інформатики та комп'ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділення
комп'ютерних технологій

Наталія СТЕФУРАК _____
(підпис)

«____» _____ 2025р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 123 «Комп'ютерна інженерія»

на тему:

«Інтелектуальна система голосового керування пристроями «розумного» дому»

Студент групи КІ-41 Олександр БЕРЕЗА _____
(підпис)

Керівник роботи Василь ПАВЛЮС _____
(підпис)

Консультанти:

з техніко-економічного обґрунтування Любов МЕЛЕНЧУК _____
(підпис)

нормоконтролер Василь КУЗИК _____
(підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В'ячеслава Чорновола
відділення комп'ютерних технологій
циклова комісія інформатики та комп'ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп'ютерних технологій
Наталія СТЕФУРАК_____
« ___» _____2024р.

ЗАВДАННЯ

на кваліфікаційну роботу
на здобуття освітньо-кваліфікаційного рівня «фаховий молодший бакалавр»
студенту
Березі Олександру Андрійовичу

1. Тема кваліфікаційної роботи: «Інтелектуальна система голосового керування пристроями «розумного» дому» затверджено наказом по коледжу Від "___" _____ 2024р., № _____
2. Термін здачі студентом завершеної роботи "___" _____ 2025р.
3. Вихідні дані до роботи: актуальні технології: технічне завдання, огляд комерційних і open-source рішень, актуальні технології голосового керування та штучного інтелекту.
4. Перелік питань, які повинні бути розроблені в роботі:

а) основна частина: аналіз предметної області та огляд наявних рішень, постановка завдання, проєктування системи, розробка архітектури системи та логіки її функціонування, вибір компонентів і програмних інструментів, тестування системи.

б) техніко-економічне обґрунтування: оцінка ринку розумних рішень, розрахунок вартості реалізації, аналіз доцільності розробки та перспектив комерціалізації.

5. Перелік графічного матеріалу: зображення інтерфейсів популярних систем, фото вибраних компонентів, структурна схема архітектури системи, блок-схеми алгоритмів роботи, візуальні схеми, схеми з'єднань компонентів, фото компонентів, фото монтажу компонентів системи, фото інтерфейсу мобільного застосунку.

6. Консультанти роботи:

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
З техніко-економічного обгрунтування	Меленчук Л.І.		

КАЛЕНДАРНИЙ ПЛАН

Виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Термін	
		початку	завершення
1.	Вибір теми, ознайомлення з вимогами до кваліфікаційної роботи	25.11.2024	07.12.2024
2.	Аналіз предметної області, вивчення наявних комерційних та open-source рішень	08.12.2024	15.01.2025
3.	Формування вимог до системи, постановка завдань	16.01.2025	28.01.2025
4.	Розробка архітектури системи, вибір технічних та програмних засобів	29.01.2025	15.02.2025
7.	Реалізація апаратної частини (монтаж плати, підключення сенсорів), розробка програмного забезпечення та мобільного застосунку	04.03.2025	28.04.2025
8.	Інтеграція підсистем, реалізація автоматизації на основі умов і сценаріїв	29.04.2025	14.05.2025
9.	Проведення функціонального тестування, усунення виявлених недоліків	15.05.2025	22.05.2025
10.	Техніко-економічне обґрунтування	23.05.2025	30.05.2025
11.	Оформлення пояснювальної записки	31.05.2025	15.06.2025
11.	Попсрсдній захист кваліфікаційної роботи	16.06.2025	16.06.2025
12.	Захист кваліфікаційної роботи	26.06.2025	26.06.2025

Дата видачі «__» _____ 2024р. Керівник _____ Василь ПАЛЮС

Завдання прийнято до виконання _____ Олександр БЕРЕЗА

Реферат

Кваліфікаційна робота. Інтелектуальна система голосового керування пристроями «розумного» дому. 143 с., 36 рисунків, 4 таблиці, 6 додатки, 16 джерел.

Метою даної роботи є розробка інтелектуальної системи для голосового керування пристроями розумного будинку на основі технологій Інтернету речей, яка забезпечує зручну та гнучку взаємодію з користувачем через мобільний застосунок і голосові команди.

Система реалізована за допомогою апаратних засобів (ESP32, ESP8266-01, датчики DHT11, MQ-9, GL5528, RGB LED стрічка, блоку живлення 12В/5А) та програмного забезпечення з використанням Flutter, Firebase, MQTT, Arduino IDE та Visual Studio Code. Застосунок дозволяє не лише керувати освітленням, а й переглядати дані сенсорів, створювати автоматизації за умовами чи часом, та отримувати голосовий зворотній зв'язок.

Особливістю системи є її гнучка архітектура з використанням протоколу MQTT, підтримка хмарної бази даних, можливість розширення сценаріїв автоматизації та інтеграція з сервісами штучного інтелекту для синтезу мовлення, розпізнавання мовлення та прийняття рішень під час обробки команд.

Під час розробки проведено тестування системи, яке підтвердило її стабільність, функціональність та відповідність вимогам. Система має потенціал для масштабування, комерційного застосування та актуальна для людей з обмеженими можливостями, оскільки дозволяє певну взаємодію без фізичних дій.

ІНТЕЛЕКТУВАЛЬНА СИСТЕМА ГОЛОСОВОГО КЕРУВАННЯ ПРИСТРОЯМИ «РОЗУМНОГО» ДОМУ, ESP32, MQTT, МОБІЛЬНИЙ ЗАСТОСУНОК, ШІ, ІНТЕРНЕТ РЕЧЕЙ.

Abstract

Qualification Work. Intelligent Voice-Controlled Smart Home Device Management System. 143 p., 36 figures, 4 tables, 6 appendices, 16 references.

The aim of this work is to develop an intelligent system for voice control of smart home devices based on Internet of Things technologies, which provides a convenient and flexible interaction with the user via a mobile application and voice commands.

The system is implemented using hardware components (ESP32, ESP8266-01, DHT11, MQ-9, GL5528 sensors, RGB LED strip, 12V/5A power supply) and software tools including Flutter, Firebase, MQTT, Arduino IDE, and Visual Studio Code. The application enables not only lighting control, but also sensor data monitoring, creation of condition- or time-based automation scenarios, and receiving voice feedback.

A key feature of the system is its flexible architecture using the MQTT protocol, cloud database support, scalability of automation scenarios, and integration with artificial intelligence services for speech synthesis, speech recognition, and decision-making in command processing.

During the development process, system testing was conducted, which confirmed its stability, functionality, and compliance with the requirements. The system has strong potential for scalability, commercial application, and is especially relevant for people with disabilities, as it enables interaction without physical contact.

INTELLIGENT VOICE-CONTROLLED SMART HOME SYSTEM, ESP32, MQTT, MOBILE APPLICATION, ARTIFICIAL INTELLIGENCE, INTERNET OF THINGS.

ЗМІСТ

					КР.КІ 25.001.01.000 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Береза О.А			Інтелектуальна система голосового керування пристроями «розумного» дому	Літ.	Арк. 5 / Аркушів 50
Перевір.		Павлюс В.П					
Реценз.		Слєпцова О.Я				ГФК. ВКТ. КІ-41	
Н. Контр.		Кузик В.М					
Затверд.		Стефурак Н.А					

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних.

ЗНГ – зріджений нафтовий газ.

ШІ – штучний інтелект.

GUI – graphical user interface (графічний інтерфейс користувача).

IoT – Internet of Things (концепція мережі, що складається із взаємозв'язаних фізичних пристроїв).

LED – light-emitting diode (світлодіод).

Wi-Fi – Wireless Fidelity (технологія бездротової мережі).

ВСТУП

Сучасні технології активно впроваджуються у побутове життя, створюючи системи автоматизації, які підвищують рівень комфорту, енергоефективності та безпеки. Одним із ключових напрямків у сфері Інтернету речей (IoT) [1] є розумні будинки, які дозволяють користувачам дистанційно контролювати різні пристрої за допомогою голосових команд, мобільних застосунків або сенсорних панелей.

Актуальність дослідження полягає у зростаючій потребі користувачів у зручних, інтуїтивних та інтелектуальних системах керування домашніми пристроями. Існуючі рішення, хоча й мають багато переваг, не завжди відповідають вимогам гнучкості, інтеграції та підтримки користувацьких запитів. Тому розробка нових підходів до автоматизації, які базуються на сучасних технологіях, є важливою складовою розвитку розумних будинків.

Метою роботи є створення інтелектуальної системи керування пристроями розумного будинку за допомогою голосових команд. Основною проблемою, що вирішується у проєкті, є забезпечення інтеграції різнорідних пристроїв та модулів у єдину систему з підтримкою інтелектуальних рішень і розширеного голосового спілкування за допомогою штучного інтелекту (ШІ).

Для досягнення поставленої мети необхідно виконати такі завдання:

1) дослідити існуючі системи керування розумним будинком, визначивши їх переваги та недоліки;

2) розробити архітектуру системи, яка включатиме центральний контролер та допоміжні модулі для забезпечення взаємодії з датчиками та актуаторами;

3) забезпечити підтримку голосових команд та інтеграцію з сервісами ШІ для розширеного спілкування з користувачем та прийняття рішень;

4) створити систему, яка підтримує гнучке масштабування та розширення, з можливістю персоналізації використання системи користувачем.

Запропонована система має потенціал для практичного застосування, оскільки дозволяє інтегрувати різні пристрої у єдину екосистему та забезпечити

більш інтуїтивний спосіб керування через голосові команди, та доступна можливість персоналізації для зручного спілкування з ШІ. Таким чином, дана розробка сприяє подальшому розвитку технологій Інтернету речей у галузі автоматизації житлових приміщень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Аналіз існуючих систем керування пристроями розумного будинку

Системи розумного будинку є комплексом програмно-апаратних рішень, які забезпечують автоматизацію та централізоване керування різними пристроями в межах житлового чи комерційного приміщення. Основними функціями таких систем є керування освітленням, кліматом, безпекою, мультимедійними пристроями та іншими елементами побутової техніки. Взаємодія з системою зазвичай здійснюється через мобільні застосунки або голосових асистентів у вигляді розумних колонок, альтернативно також може бути доступне керування через веб-інтерфейси, що дозволяє реалізувати дистанційне управління пристроями в режимі реального часу.

Проаналізувавши існуючі системи керування розумним будинком, розглянемо три найпопулярніші: Google Home, Amazon Alexa та Home Assistant. Кожна з цих платформ має власну архітектуру, механізм взаємодії з користувачем та рівень персоналізації.

1.1.1 Google Home

Google Home [2] (рис. 1.2) – це екосистема розумного будинку, що базується на Google Assistant, який є центральним елементом керування. Архітектура платформи складається з серверної частини, що забезпечує обробку голосових запитів у хмарі, а також локальних пристроїв, які взаємодіють через протоколи Wi-Fi, Bluetooth та Thread. Інтеграція нових пристроїв здійснюється через Google Home App (рис. 1.1), де користувач може додавати та налаштовувати сумісне обладнання, а також створювати автоматизації та керувати всіма підключеними елементами розумного будинку.

Протокол Wi-Fi використовується для підключення більшості розумних пристроїв, оскільки забезпечує високу швидкість передачі даних і стабільність з'єднання. Bluetooth застосовується для сполучення пристроїв на ближчих відстанях, таких як динаміки або сенсори, що працюють у локальній мережі.

Thread є відносно новим протоколом, який забезпечує енергоефективну бездротову комунікацію між пристроями розумного будинку, дозволяючи їм працювати у відмовостійкій mesh-мережі без потреби у центральному хабі.

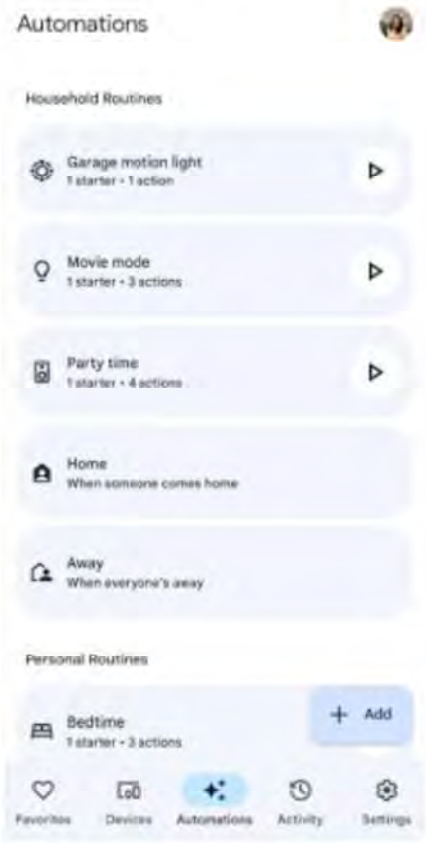

Рисунок 1.1 – Інтерфейс додатку Google Home App

Ключовою особливістю Google Home є його здатність до розпізнавання голосових команд. Платформа використовує алгоритми машинного навчання для ідентифікації мови, аналізу запитів та виконання відповідних дій. Запити користувачів надсилаються на сервери Google, де відбувається їх обробка, після чого команди повертаються на відповідні пристрої.

Система дозволяє створювати сценарії автоматизації, використовуючи Google Routines – механізм, який дає змогу запускати кілька дій за однією командою. Наприклад, при команді «Добрий ранок» Google Home може вмикати світло, запускати кавоварку та програвати новини.

Проте, однією з головних проблем платформи є обмежена підтримка нестандартних пристроїв. Google Home здатний працювати лише з

сертифікованими пристроями, що можуть інтегруватися через Google Assistant API. Крім того, частина функцій доступна лише за передплатою у Google One або в преміум-версії деяких додатків.

Рисунок 1.2 – Платформа Google Home

1.1.2 Amazon Alexa

Amazon Alexa [3] (рис. 1.4) є однією з найбільш розвинених платформ для керування розумним будинком. Архітектура Alexa базується на взаємодії між пристроями через «Alexa Skills» – спеціальні розширення, які додають нові функціональні можливості. Взаємодія з користувачем відбувається через голосові команди, які передаються на сервери Amazon для обробки та генерації відповідних відповідей.

Головною перевагою Amazon Alexa є її висока інтеграційна здатність. Платформа підтримує широкий спектр пристроїв, які можуть комунікувати через протоколи Wi-Fi, Zigbee та Z-Wave. Крім того, Alexa дозволяє налаштовувати складні сценарії автоматизації через «Alexa Routines».

Комунікація між пристроями здійснюється через такі підтримувані протоколи:

– MQTT (Message Queuing Telemetry Transport) – легковаговий протокол, оптимізований для передавання даних у реальному часі між пристроями IoT.

– Zigbee – бездротовий протокол низького енергоспоживання, що дозволяє пристроям комунікувати у mesh-мережі, забезпечуючи надійний зв'язок.

– Z-Wave – подібний до Zigbee, але працює на іншій частоті і зазвичай використовується у рішеннях для автоматизації будинку.

Інтерфейс додатку Amazon Alexa зображений на рисунку 1.3.

Розпізнавання голосу у Amazon Alexa базується на нейромережах, що дозволяють асистенту адаптуватися до голосу користувача та розрізняти різних членів родини. Це забезпечує персоналізований досвід взаємодії, де кожен користувач може отримувати унікальні відповіді та автоматизації відповідно до власних уподобань. Голосові запити проходять через багаторівневу систему обробки, що складається з модулів розпізнавання мови (ASR – Automatic Speech Recognition), природного розуміння мови (NLU – Natural Language Understanding) та генерації відповіді (TTS – Text-to-Speech). Моделі ASR аналізують мовлення, перетворюючи його у текст, а NLU визначає намір користувача та пов'язує його із відповідною командою або навичкою.

Alexa використовує рекурентні нейромережі (RNN) та трансформерні архітектури (BERT, GPT-подібні моделі) для покращення точності розпізнавання та аналізу контексту діалогу. Це дозволяє асистенту не лише розпізнавати команду, а й враховувати історію взаємодій для коригування відповідей. Крім того, Alexa підтримує самонавчання, адаптуючись до особливостей голосу кожного користувача, що підвищує точність команд і зменшує ймовірність помилкових відповідей.

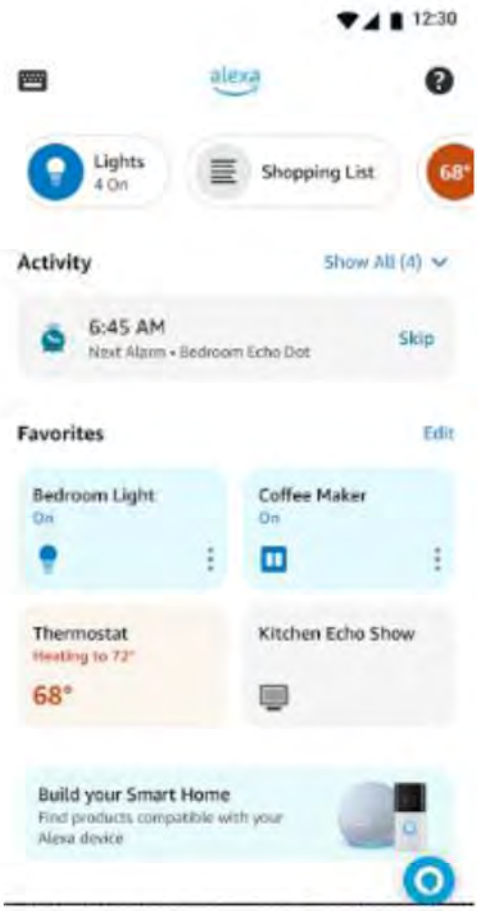

Рисунок 1.3 – Інтерфейс додатку Amazon Alexa

Проте, як і Google Home, Alexa залежить від хмарних сервісів, а також має певні обмеження у локальному керуванні пристроями без підключення до інтернету. Частина функцій, таких як Alexa Guard (система безпеки), доступна лише в платній підписці Alexa Guard Plus. Крім того, додавання сторонніх пристроїв може вимагати складної конфігурації через «Alexa Skills», що робить її менш зручною для користувачів без технічних знань.

amazon alexa

Рисунок 1.4 – Платформа Amazon Alexa

1.1.3 Home Assistant

Home Assistant [4] (рис. 1.6) – це платформа з відкритим кодом, що дозволяє створювати повністю автономні системи керування розумним будинком. Вона працює за принципом локального зберігання даних і не потребує обов'язкового підключення до хмарних сервісів, що робить її більш захищеною з точки зору конфіденційності. Інтерфейс зображено на рисунку 1.5.

Архітектура Home Assistant базується на сервері, який можна розгорнути на Raspberry Pi або іншому локальному сервері. Raspberry Pi – це мікрокомп'ютер, який широко використовується для створення персоналізованих IoT-рішень, зокрема для розгортання серверних платформ автоматизації.

Комунікація між пристроями також здійснюється через MQTT, Zigbee, Z-Wave або локальний Wi-Fi. Основною перевагою платформи стала її гнучкість: користувач може інтегрувати будь-який пристрій, навіть якщо він офіційно не підтримується іншими платформами. Саме це зробило дану платформу настільки популярною серед ентузіастів, які реалізовують персоналізовані системи розумного дому.

Розпізнавання голосових команд у Home Assistant реалізується через додаткові модулі, такі як Rhasspy або Almond. Rhasspy – це автономна голосова система, що дозволяє здійснювати розпізнавання мови локально без підключення до хмарних сервісів. Almond – це голосовий асистент з відкритим кодом, що може виконувати команди користувача, базуючись на локальній обробці запитів.

Рисунок 1.5 – Інтерфейс додатку Home Assistant

Попри свої переваги, Home Assistant має високий поріг входу для новачків. Налаштування платформи потребує глибокого розуміння систем автоматизації, роботи з конфігураційними файлами та написання YAML-скриптів. Крім того, оновлення можуть викликати конфлікти у системі, що вимагає додаткової технічної підтримки з боку користувача.

Рисунок 1.6 – Платформа Home Assistant

Аналіз існуючих рішень дав зрозуміти, що кожна платформа має свої особливості та недоліки. Google Home та Amazon Alexa пропонують користувачам простоту використання та розширену екосистему підтримуваних пристроїв, але мають обмежену автономність та високу залежність від хмарних сервісів. Home Assistant, у свою чергу, забезпечує максимальну гнучкість та

безпеку, проте його налаштування може бути складним для недосвідчених користувачів.

1.2 Постановка завдань для розробки системи

Зважаючи на особливості сучасних платформ керування розумним будинком, можна зробити висновок, що кожна з них має свої переваги та обмеження. Деякі рішення надають простоту використання та широкий спектр сумісних пристроїв, інші орієнтовані на гнучкість і повний контроль з боку користувача. Таким чином, для розробки власної системи керування пристроями розумного будинку необхідно взяти до уваги найкращі практики кожної з платформ, об'єднавши простоту використання, гнучкість налаштувань та можливість автономної роботи без залежності від хмарних сервісів.

Одним із ключових аспектів є голосове керування пристроями. Інтеграція з хмарними сервісами дозволяє обробляти складні голосові команди з достатньою точністю і допустимою похибкою розпізнавання намірів користувача та здійснювати персоналізовану взаємодію. Використання штучного інтелекту дасть змогу розширити можливості керування, додаючи аналіз контексту запитів та створення адаптивних сценаріїв автоматизації. Також важливим елементом є безпека та контроль доступу, що дозволить убезпечити дані користувачів і запобігти несанкціонованому втручанню.

Для реалізації ефективного зв'язку між пристроями система має підтримувати сучасні протоколи бездротової комунікації. Використання Wi-Fi забезпечить швидку передачу даних між вузлами системи та надасть можливість централізованого контролю. Bluetooth Low Energy (надалі – BLE) можна буде розглядати як альтернатвне рішення для подального розвитку системи, яке стане в нагоді для сенсорних модулів, що працюють на низькому енергоспоживанні. Важливим є впровадження надійних протоколів обміну даними. MQTT, як один із провідних стандартів у сфері інтернету речей, дозволить зменшити затримки в передачі команд, водночас додаючи можливість швидкої інтеграції інших пристроїв та оптимізувати навантаження на мережу.

Гнучка інтеграція різних типів пристроїв дасть змогу користувачам легко додавати нові модулі, що робить систему масштабованою. Додатково, REST API, тобто архітектурний стиль взаємодії між клієнтом і сервером, що використовує протокол HTTP для виконання запитів на отримання або зміну даних дозволить інтегрувати систему з іншими сервісами, надаючи можливість створення складніших сценаріїв автоматизації.

Розпізнавання голосу відіграватиме центральну роль у функціонуванні системи. Обробка команд потребує використання нейромережевих моделей, які забезпечать точне розпізнавання мовлення, проте розгортання локальної нейромережної моделі обробки голосу потребує значних апаратних можливостей та великої кількості часу для навчання моделі, через що доцільним буде використання хмарних сервісів з готовими нейромережевими моделями для обробки голосу, що дозволить досягти оптимальної якості з мінімальними похибками за рахунок невеликих затримок, аналогічно застосування технологій синтезу мовлення дасть змогу забезпечити голосові відповіді системи, створюючи більш природну взаємодію з користувачем. Використання рекурентних нейромереж і трансформерних архітектур дозволить покращити якість обробки мовних команд.

Збереження та обробка даних є критичним компонентом будь-якої розумної системи. Для цього необхідно використовувати сучасні бази даних, які забезпечать ефективне зберігання та обробку інформації. Хмарні рішення, такі як Firebase, дозволять синхронізувати дані в реальному часі між усіма потрібними пристроями та зберігати дані віддалено, що допоможе створити гнучку архітектуру для керування інформацією про пристрої, історію взаємодії та параметри користувачів. Додатково, локальне кешування даних забезпечить автономність роботи системи в разі тимчасового відключення Інтернету.

Розробка даної системи дозволить створити ефективне рішення для керування пристроями розумного будинку, що забезпечить гнучкість, безпеку та масштабованість. Поєднання сучасних технологій розпізнавання голосу,

штучного інтелекту та пристроїв Інтернету речей дасть змогу інтегрувати систему з широким спектром пристроїв, забезпечуючи повноцінний користувацький досвід. Гнучкість архітектури та підтримка протоколу зв'язку MQTT зробить систему універсальною для інтеграції з різними розумними пристроями.

Таким чином, запропонована система не лише створить зручне середовище для автоматизації житлових приміщень, а й закладе основу для подальшого розвитку розумних екосистем. Для виконання завдання можна Подальше вдосконалення може включати інтеграцію додаткових модулів, застосування алгоритмів машинного навчання для прогнозування поведінки користувачів та розширення можливостей самонавчання системи.

Зробивши підсумки, система повинна виконувати такі функціональні та нефункціональні вимоги:

– використання ШІ для реалізації обробки рішень;

– голосове керування та озвучення відповіді системи через зв'язок із хмарними сервісами;

– можливість керування пристроями та моніторингу їх даних у застосунку;

– забезпечення зберігання даних пристроїв та історії комунікацій у хмарній БД;

– масштабованість та гнучкість за рахунок використання популярного протоколу комунікації MQTT;

– компактність та зручність інтерфейсу застосунку;

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування архітектури системи

Інтелектуальна система керування пристроями розумного будинку повинна бути реалізована як сукупність взаємопов'язаних підсистем, у центрі якої знаходиться мобільний застосунок. Саме він виконуватиме ключову функцію керування: через нього користувач ініціюватиме дії, налаштовуватиме сценарії автоматизації, а також отримуватиме зворотній зв'язок з пристроїв системи.

Комунікацію між застосунком і фізичними пристроями доцільно буде реалізувати за допомогою брокера MQTT. Всі команди, сформовані користувачем, надсилатимуться в відповідні MQTT-топіки для відповідних виконавчих модулів. Зокрема, головна плата на отримуватиме ці повідомлення, інтерпретуватиме їх, порівнюючи з поточним контекстом (наприклад, часом чи умовами довкілля), і виконуватиме відповідну дію, наприклад, зміну кольору освітлення.

Дані з сенсорів, що розміщені на допоміжних модулях, також передаватимуться до MQTT-брокера. Мобільний застосунок зчитуватиме ці дані, виводячи їх у зручному вигляді на інтерфейсі, а також синхронізуватиме з хмарною БД для зберігання історії.

Головна плата також зчитуватиме дані отримані із сенсорних вузлів для того щоб керувати відповідними процесами без необхідності використання застосунку за допомогою попередньо створених сценаріїв або запланованих подій, завдяки цьому система зможе підтримувати функціонал автоматичного керування. Користувач зможе задати умову на кшталт: «якщо температура перевищує 25°C — змінити колір освітлення на синій». Такі правила будуть формуватись в застосунку, надсилатись до MQTT брокера, і надалі оброблятимуться головною платою. Також користувач матиме змогу вказати точну дату та час для виконання певної дії, після чого головна плата самостійно перевірятиме поточні дату та час і виконуватиме команду у визначений момент.

Планування команд можна також зробити повторюваними, щоб наприклад кожного дня, або якогось певного дня тижня, о 10:00 годині дня включалось світло теплого відтінку або озвучувалась поточна погода.

Таким чином, мобільний застосунок виконуватиме роль «мозку» системи, який управлятиме всією логікою, тоді як апаратні пристрої – це виконавці, що будуть зчитувати, реагувати та виконувати відповідні дії.

2.2 Проєктування алгоритму роботи системи

Основу логічної структури складатиме тісна зв'язка між мобільним застосунком, MQTT-брокером та виконавчими пристроями. Користувач взаємодіятиме із системою виключно через застосунок, у якому буде реалізовано інтерфейс для відображення даних з сенсорних модулів та погоду, панель для керування освітленням, а також панель для налаштування сценаріїв автоматизації на основі правил або часу.

Коли користувач надсилатиме команду, вона передаватиметься через MQTT. Наприклад, команда «увімкнути червоне освітлення о 20:00» буде збережена в відповідному MQTT-топіку, з якого головна плата її зчитає та відкладе до внутрішнього буфера подій. Далі контролер постійно порівнюватиме системний час з часом запланованих подій і виконає дію, коли настане відповідний момент.

Для автоматизації за умовами система працюватиме за аналогічним принципом. Користувач задаватиме правило, наприклад: «якщо температура менше 18°C – ввімкнути зелений колір світла». Це правило надсилатиметься через MQTT, зберігаючись в відповідному MQTT-топіку, і перевірятиметься щоразу, коли надходять нові дані з сенсорів. Якщо умова виконується, плата виконуватиме закладену дію.

Мобільний застосунок не тільки дозволятиме створювати команди, але й зчитувати MQTT-повідомлення від вузлів системи з показниками температури, вологості, присутності шкідливих газів у повітрі всередині приміщення, та також зчитувати інформацію про зовнішні погодні умови із онлайн сервісу через HTTP-

запити (REST API). Дані виводитимуться у застосунку та автоматично зберігатимуться у хмарну БД. Таким чином, буде реалізована не лише взаємодія з пристроями в реальному часі, а й фіксація історії для подальшого аналізу або побудови статистики.

Весь обмін даними між компонентами має бути побудований асинхронно, що дозволить системі працювати стабільно навіть у випадках короткочасної втрати зв'язку з брокером або мережею. Для кращого розуміння функціонування системи було спроектовано та створено блок-схеми алгоритмів в хмарному середовищі Draw.io (дод. А рис. 1-3).

У системі був обраний підхід, де логіка та ініціатива будуть зосереджені у мобільному застосунку, оскільки це даватиме користувачу максимальний контроль та гнучкість. Така архітектура дозволить легко додавати нові функції без втручання у прошивку апаратних модулів.

MQTT-протокол було обрано через його ефективність у ресурсно обмежених системах, низькі затримки, підтримку широкого кола пристроїв та зручність при реалізації publish/subscribe-моделі. Це забезпечить надійний двосторонній зв’язок між пристроями й застосунком.

Основою центрального модуля має бути плата з наявністю вбудованого Wi-Fi та можливостями керування GPIO-портами, також важливо щоб плата мала можливість керувати ШІМ сигналами, які будуть необхідні для керування каналами кольорів світла. Водночас для основи сенсорних модулів потрібно обрати плати, які будуть компактними та енергоефективними, оскільки сенсорні модулі повинні мати можливість вільного розміщення будь-де у приміщенні, без прив’язки до місця розташування головної плати.

Автоматизація за правилами та календарем має бути реалізована без необхідності підключення до зовнішніх серверів, що підвищить автономність та надійність системи. Дані з сенсорів та вся інформація про дії системи зберігатимуться у хмарній БД, що дозволить легко реалізувати інтерфейс історії,

звітності або аналітики та водночас забезпечить доступ до даних із будь-якого пристрою.

Таким чином, система була спроєктована як модульна, гнучка та розширювана, із застосуванням сучасних підходів до автоматизації на основі часу та умов. Центральну роль у ній відіграватиме мобільний застосунок, який забезпечуватиме повноцінне керування пристроями, налаштування сценаріїв, перегляд даних і зворотній зв'язок. У межах розділу розроблено архітектуру, описано алгоритм функціонування з використанням MQTT-протоколу та наведено логіку взаємодії з сенсорами. Обґрунтовано вибір технічних рішень і підкреслено переваги асинхронної взаємодії між компонентами. Запропонована структура спрямована на стабільність, автономність та масштабованість у реальних умовах використання.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Реалізація програмно-технічного засобу передбачає створення повнофункціонального, надійного та стабільно працюючого пристрою на основі розробленого проєкту «Інтелектуальна система голосового керування пристроями «розумного» дому». Основна мета цього етапу — практичне втілення технічного задуму у вигляді прототипу, здатного реагувати на сенсорні дані, виконувати команди з мобільного застосунку, а також підтримувати автоматизацію дій залежно від заданих умов.

3.1 Вибір компонентів системи

Для успішної реалізації проекту було обрано сучасні апаратні компоненти, що забезпечують стабільність, ефективність та зручність інтеграції. Вартість цих обраних компонентів вважається відносно низькою, що робить їх бездоганним вибором для створення дешевої та водночас хорошої альтернативи до вже існуючих систем керування пристроями розумного дому.

Ключовим елементом системи стала плата ESP32 DevKit V1 [5] (рис. 3.1), яка має необхідну обчислювальну потужність і можливості підключення до Wi-Fi.

Рисунок 3.1 – Плата ESP32 DevKit V1

Модуль розробника побудований на мікромодулі ESP-WROOM-32, який поєднує модулі Wi-Fi, Bluetooth і BLE, що забезпечує універсальність і

масштабованість системи. Обрання саме цієї плати обґрунтоване такими перевагами:

– висока швидкість передачі даних (до 150 Мбіт/с);

– підтримка різних режимів роботи Wi-Fi (STA/AP/STA + AP/P2P);

– можливість роботи в умовах широкого діапазону температур (-40 - +125 °C);

– енергозбереження (споживання у режимі глибокого сну до 20 мкА).

Плата також має розширені функції Bluetooth для роботи з пристроями ближнього зв'язку. Дані переваги роблять її оптимальним вибором для центрального контролера системи, що забезпечує стабільне підключення і обробку команд.

Для реалізації допоміжних модулів з датчиками або актуаторами було обрано ESP8266-01 [6] (рис. 3.2).

Рисунок 3.2 – Плата ESP8266-01

Цей компактний і енергоефективний модуль забезпечує можливість роботи в умовах обмежених ресурсів. Його ключові характеристики:

– підтримка Wi-Fi стандарту 802.11 b/g/n;

– низьке енергоспоживання (3.3В, до 240мА), що дозволяє використовувати невеликі акумулятори або батарейки як джерела живлення;

– простота інтеграції з різними сенсорами та виконавчими пристроями.

Модуль ESP8266-01 стане основою для створення окремих автономних вузлів, які взаємодіятимуть із центральним контролером через бездротовий зв'язок.

RGB LED стрічка довжиною 5 метрів [7] буде використовуватись як джерело освітлення яким можна буде керувати використовуючи ШІМ сигнали (рис. 3.3).

Рисунок 3.3 – Світлодіодна RGB LED стрічка

LED стрічка RGB 12V SMD5050 30LED/m ip20 - це тонка та гнучка стрічка зі світлодіодами, рівномірно на ній розташованими, призначена для декоративної підсвітки. Кожен метр стрічки містить 30 RGB світлодіодів SMD5050, що забезпечують помірно яскраве і рівномірне світло.

Характеристики:

- робоча напруга: 12 В;
- потужність: 7,2 Вт\м;
- кут розсіювання: 120°;
- клас захисту: IP20 (негерметична).

Ці характеристики роблять дану світлодіодну стрічку хорошим варіантом, оскільки споживання відносно невелике в порівнянні з іншими LED стрічками, але водночас хороша якість освітлення, що дозволить зменшити витрати на електроенергію без зміни якості освітлення.

Блок живлення з потужністю 75 Вт [8] забезпечить надійне і стабільне живлення для головної плати та світлодіодної стрічки (рис. 3.4).

Рисунок 3.4 – Блок живлення 12В.

Цей компонент було обрано враховуючи споживчу потужність плати ESP32 та світлодіодної стрічки, так як він буде виконувати подвійну функцію: безпосередньо забезпечуватиме живленням RGB LED стрічку, та через стабілізатори напруги забезпечуватиме живлення плати ESP32.

Характеристики:

- вихідна напруга 12В;
- максимальний вихідний струм 5А;
- світлодіод показує стан блоку живлення.

Характеристики даного компонента дозволяють використовувати його на постійній основі як джерело живлення для основної плати та світлодіодної стрічки, без ризиків перепаду напруги.

Параметричний стабілізатор напруги 5В LM7805 [9] (рис. 3.5).

Рисунок 3.5 – Стабілізатор напруги 5В LM7805 в корпусі TO-220

Цей компонент буде використовуватись для забезпечення стабільного живлення яке не перевищуватиме 5В, що є критичним для надійної роботи плати. Стабілізатор незамінний в ситуаціях коли важлива не тільки стабільність напруги, але і мінімальний рівень шумів напруги живлення. Просте підключення та мінімальна кількість додаткових зовнішніх компонентів.

Характеристики:

- вхідна напруга: +7.5 ... +25 В;
- вихідна напруга: 5.0 В;
- максимальний вихідний струм: 1.0 А (при використанні радіатора);
- корпус: TO-220.

Для керування каналами світла було обрано силовий польовий транзистор IRLZ44N [10] (рис. 3.6).

Рисунок 3.6 – Силовий польовий транзистор IRLZ44N

Особливістю цього MOSFET транзистора є низька напруга Затвор-Стік, необхідна для достатнього відкривання транзистора. Ця особливість дозволяє використовувати його без додаткових ключів з безпосереднім управлінням від логічних схем з напругою 3.3-5В. Виконаний в корпусі TO-220AB, при великих струмах кріпиться на радіатор, між витоком і стоком є вбудований захисний діод. Доступна вартість транзистора сприяють його широкому поширенню в побутовій та промисловій електроніці. Має дуже низький опір, швидке переключення та малий заряд затвора.

Характеристики:

- опір «витік-стік» (у відкритому стані) 30 мОм ;
- макс. розсіювана потужність 110 Вт ;

– напруга пробою затвор-витік – 16В;

– заряд затвора – 48нКл;

– макс. допустимий постійний струм стоку – 33А (при 100 °С).

Постійний резистор на 220Ом (рис. 3.7), який буде підтримувати довговічність системи та використовуватись для захисту плати від залишкового струму транзисторів.

Рисунок 3.7 – Постійний резистор на 220Ом

Характеристики:

– тип: метало плівковий;

– потужність: 0.25Вт;

– номінал: 220Ом.

Гніздо живлення DC 5.5/2.5мм, з клемною колодкою для зручності подачі живлення до плати (рис. 3.8)

Рисунок 3.8 – Гніздо живлення 5.5/2.5мм

Проводи фіксуються гвинтовою клемою, що забезпечує швидкий монтаж і надійне з'єднання проводів без пайки. Для зручності монтажу є розмітка полярності контакту. Характеристики:

– тип роз'єму: DC Jack 5.5 x 2.5 2P під гвинт;

– робоча напруга: до 36 В;

– максимально допустимий струм: 3A-36V / 5A-24V;

– розміри: 38 x 14 x 10 мм;

– ізоляція: ABS пластик.

DHT11 [11] (рис. 3.9), буде корисним навантаженням та важливою частиною допоміжних модулів системи.

Рисунок 3.9 – Модуль датчика вологості та температури DHT11

Даний компонент це цифровий датчик температури та вологості, що дозволяє з легкістю вимірювати показники температури та вологості повітря з частотою не більше 1Гц, основні характеристики:

– живлення: 3.5-5.5 В;

– визначення вологості: 20 - 90% RH ± 5% (макс.)

– визначення температури: 0 ~ +50 ºC ± 2% (макс.)

Модуль датчика MQ-9 [12] (рис. 3.10) дозволить спостерігати інформацію про присутність шкідливих газів у повітрі в реальному часі.

Рисунок 3.10 – Модуль датчика MQ-9

Датчик газу на базі датчика MQ-9, виконаний у вигляді модуля (на платі з необхідним обвісом), використовується для перевірки витоків газу в домашніх і промислових умовах. Реагує на ЗНГ, ізобутан, метан, алкоголь, гідроген, дим і т.п. Чутливість можна регулювати потенціометром (вже встановлено на платі). Основні характеристики:

– аналоговий вихід датчика 0 - 5В (чим вище концентрація газів тим вище напруга на виході);

– напруга живлення модуля: 5В постійного струму;

Типові діапазони вимірювань:

– метан та пропан-бутан: від 500ppm до 10000ppm;

– CO: від 20ppm до 2000ppm.

Модуль датчика освітлення для реалізації автоматичного режиму освітлення, та зчитування кількості світла (рис. 3.11).

Рисунок 3.11 – Модуль датчика освітлення

Аналогово-цифровий модуль датчика освітленості на основі фоторезистора GL5528 [13] . Поріг чутливості датчика можна встановлювати вбудованим потенціометром у широких межах – від дуже слабкого фонового світла і до спрацьовування лише на прямі сонячні промені. Основні характеристики:

– напруга живлення: 3.3-5В;

– вихідний сигнал: аналоговий та цифровий;

Для забезпечення живлення сенсорних модулів було обрано батарейки AAA [14] (рис. 3.12).

Рисунок 3.12 – Батарейки AAA для живлення сенсорних модулів

Дані батарейки мають такі характеристики:

– тип батареї: лужні;

– напруга: 1.5 В;

– ємність: 1200 мА/г.

Для зручності подачі живлення до сенсорних модулів було також обрано батарейний відсік на дві батарейки AAA [15] (рис. 3.13).

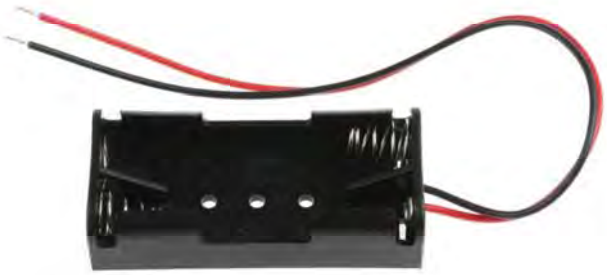

Рисунок 3.13 – Батарейний відсік

На рисунку 3.14 подано структурну схему інтелектуальної системи голосового керування пристроями розумного дому. Вона ілюструє зв'язки між основними модулями: центральним контролером, допоміжними модулями, хмарними сервісами, мобільним застосунком та брокером MQTT.

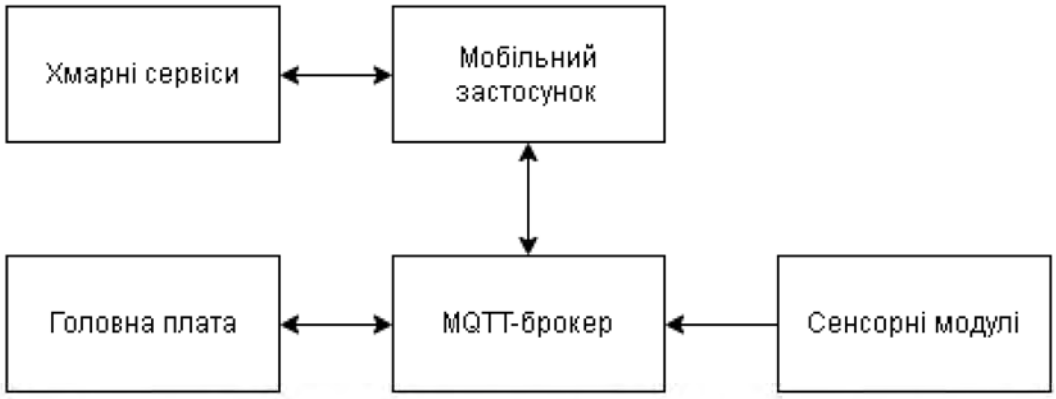

Рисунок 3.14 – Структурна схема інтелектуальної системи

3.2 Реалізація системи

У ході розробки було обрано:

– ESP32 – як головний обчислювальний модуль, що приймає MQTT-команди, керує світлодіодною стрічкою, зчитує дані з датчика MQ-9, обробляє сценарії автоматизації за часом і умовами.

– ESP8266-01 – як основа для допоміжних сенсорних модулів, що передають показники температури, вологості та освітленості через MQTT.

– Flutter і Dart – для створення мобільного застосунку з графічним інтерфейсом для платформи Android, що приймає/надсилає дані до MQTT-брокера, отримує інформацію від сенсорів та OpenWeatherMap, синхронізує її з Firebase та дозволяє формувати сценарії автоматизації.

– Firebase – як хмарну БД для зберігання історії подій.

– MQTT-брокер – як основа для обміну повідомленнями між усіма компонентами системи.

В процесі реалізації інтелектуальної системи використовувались такі середовища:

– Fritzing – принципово-електричні та візуальні схеми.

– Arduino IDE – прошивання контролерів.

– Visual Studio Code – реалізація мобільного застосунку.

Було реалізовано:

Візуальну схему підключення компонентів до головної плати системи (рис. 3.15) та візуальні схеми підключення сенсорів та живлення до сенсорних модулів, що зображені на рисунках 3.16 і 3.17.

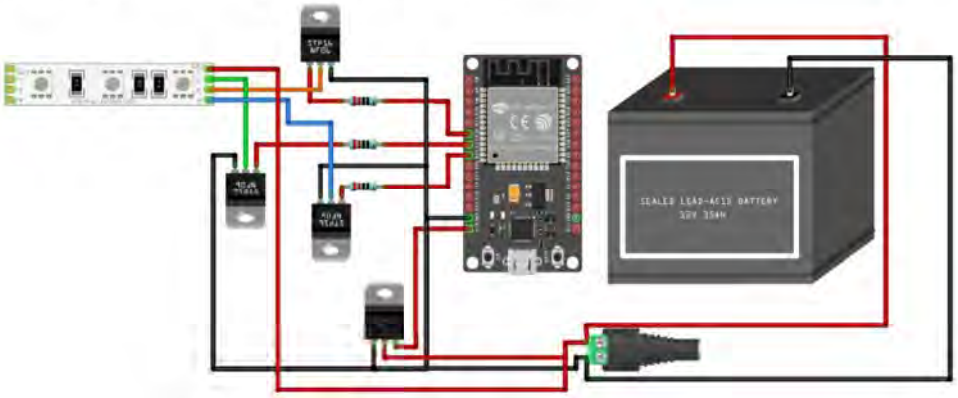

Рисунок 3.15 – Візуальна схема підключення компонентів до головної плати системи

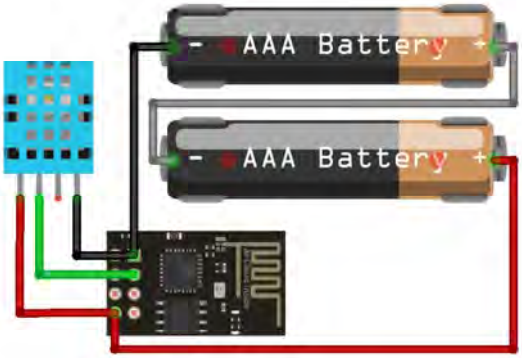

Рисунок 3.16 – Візуальна схема підключення сенсорного модуля з DHT11

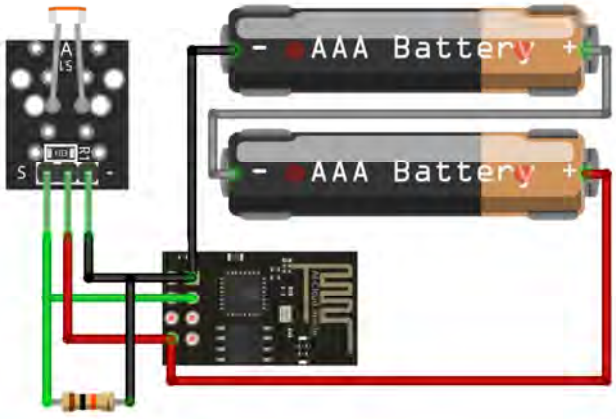

Рисунок 3.17 – Візуальна схема підключення сенсорного модуля з GL5528

Принципово-електричну схему головного модуля на ESP32 з живленням від блока живлення, транзисторами IRLZ44N для керування каналами RGB LED стрічки та модулем датчика MQ-9, що зображена в додатку Б на рисунку 1. Принципово-електричні схеми для допоміжних сенсорних модулів: з DHT11і GL5528 на базі ESP8266-01, що зображені в додатку Б на рисунках 2-3.

Під час складання принципово-електричної схеми підключення компонентів до головної плати, було відведено такі піни для контролю LED стрічкою:

– транзистор, що відповідатиме за червоний канал LED стрічки буде приєднано до контролюючого піна 32 (GPIO32);

– транзистор, що відповідатиме за зелений канал LED стрічки буде приєднано до контролюючого піна 33 (GPIO33);

– транзистор, що відповідатиме за синій канал LED стрічки буде приєднано до контролюючого піна 25 (GPIO25);

Ці піни було обрано тому, що через них можна передавати ШІМ сигнал. Програма контролера написана в середовищі Arduino IDE, лістинг програмного коду якої знаходиться в додатку В. Вона містить модулі:

– обробки MQTT-повідомлень;

– виконання умовних сценаріїв (температура, освітленість);

– виконання планування дій за часом.

Програми контролерів для допоміжних сенсорних модулів також написані в середовищі Arduino IDE, лістинг програмного коду яких знаходиться в додатку Г та додатку Д. Вони містять модулі для зчитування даних із одного приєднаного сенсора та відправлення повідомлень до MQTT-брокера.

Після прошивання кожної плати, було виконано монтаж компонентів, що зображений на рисунку 3.18.

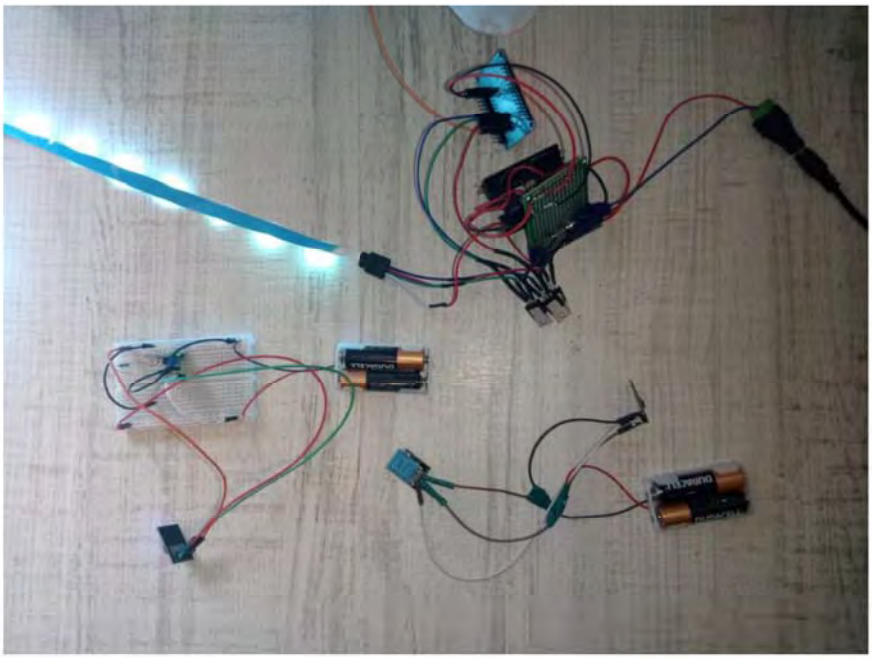

Рисунок 3.18 – Монтаж компонентів системи

Після завершення монтажу пристроїв, наступним етапом реалізації стала розробка мобільного застосунку. Програмний код якого міститься в додатку Ґ. Інтерфейс мобільного застосунку передбачає:

1. Панель керування освітленням (рис. 3.19).

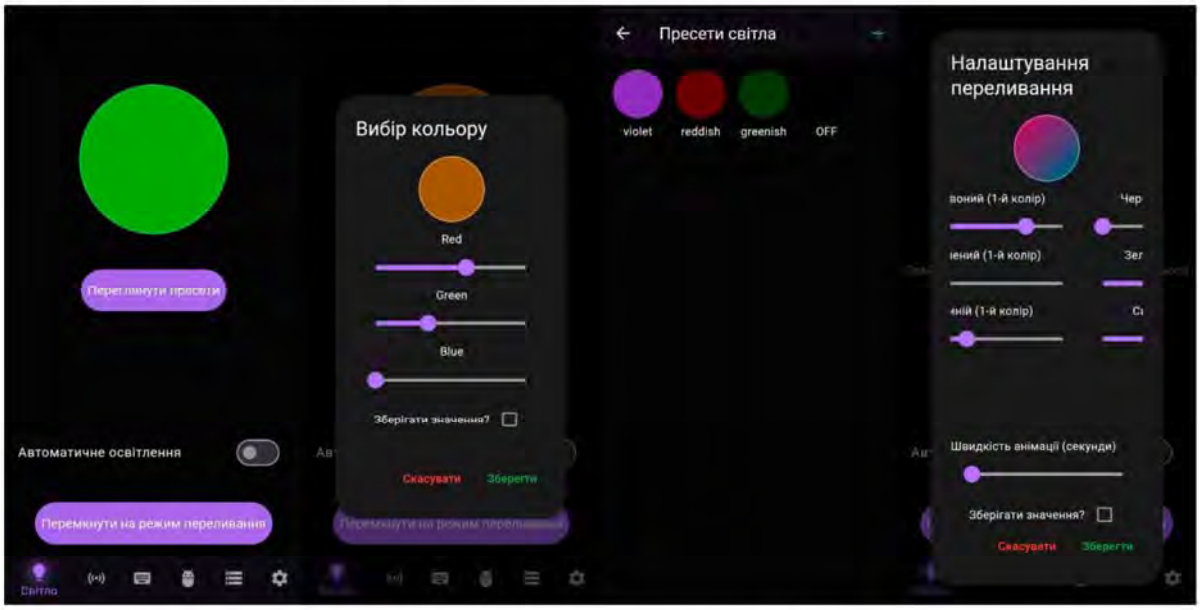

Рисунок 3.19 – Панель керування освітленням

2. Вивід сенсорних даних (температура, вологість, гази, освітленість) та даних навколишнього середовища (рис. 3.20).

Рисунок 3.20 – Сторінка із даними сенсорів та навколишнього середовища

3. Сторінку для керування за допомогою текстових та голосових запитів (рис. 3.21).

Рисунок 3.21 – Сторінка для керування через текстові та голосові запити

4. Вікно налаштування автоматизації (за часом або умовами) (рис. 3.22).

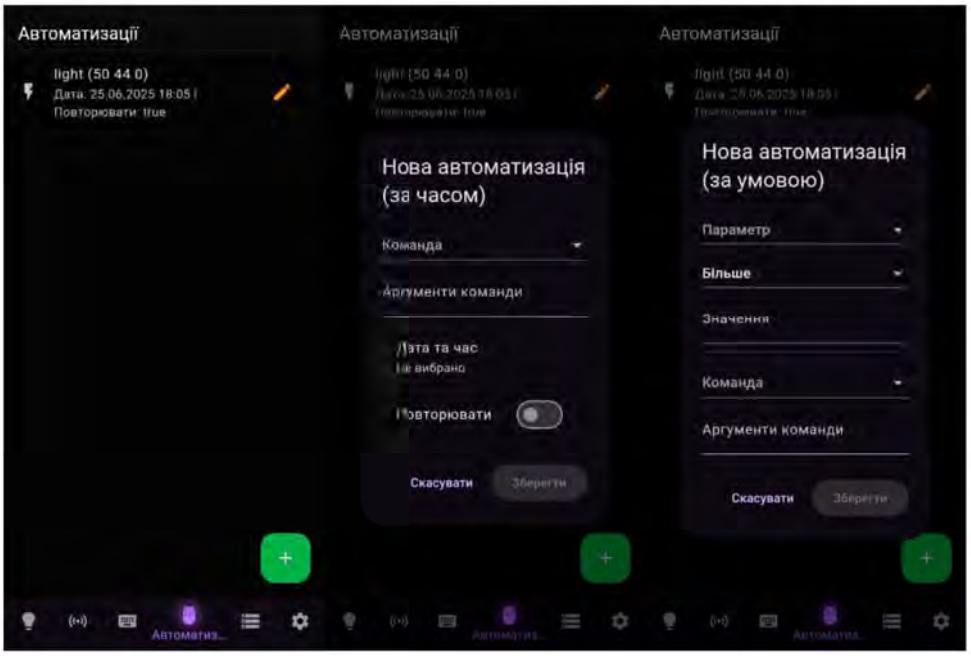

Рисунок 3.22 – Сторінка з налаштуваннями автоматизації

5. Історію подій (через Firebase) (рис. 3.23).

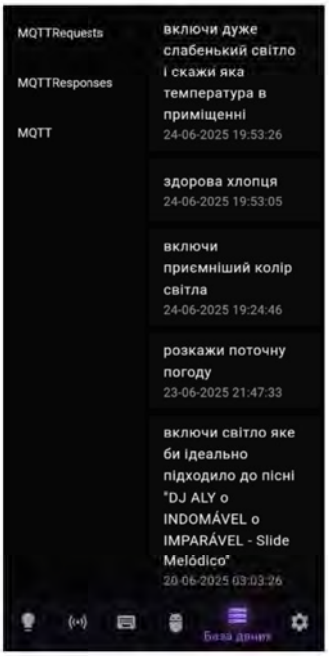

Рисунок 3.23 – Сторінка для перегляду історії подій

6. Сторінку з налаштуваннями користувача (рис. 3.24).

Рисунок 3.24 – Сторінка з налаштуваннями користувача

3.3 Тестування роботи системи

Було обрано такі методи тестування:

– «Біла скринька» – для модулів ESP32/ESP8266-01, тестування логіки сценаріїв.

– «Чорна скринька» – для мобільного застосунку: тестування GUI та поведінки.

– Функціональне тестування – відповідність системи завданням.

Тест-кейси, що застосовувались для проведення тестування системи наведені в таблиці 3.1.

Таблиця 3.1 – Тест-кейси проведення тестування системи

ID	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Увімкнення червоного світла о 20:00	Команда в застосунку	Вмикається червоне освітлення о 20:00	Ввімкнулося червоне освітлення в 20:00	Успіх
2	Зміна кольору світла на червоний за збільшення температури	Темп. > 25°C	Світло змінюється на червоне	Світло змінилося на червоне	Успіх
3	Надходження даних з MQ-9	Показники сенсора	Дані оновлюються в застосунку	Дані оновились	Успіх
4	Збереження історії у Firebase	Будь-які події	Дані записуються	Дані записано	Успіх
5	Відображення даних з OpenWeatherMap API	REST-запит	Актуальні дані про погоду	Дані відображено	Успіх

Також для перевірки базового функціоналу системи було проведено тестування таких функцій:

– Зміна кольору світла вручну користувачем у застосунку – колір світла RGB LED стрічки успішно змінюється на заданий користувачем.

– Зміна режиму освітлення користувачем в застосунку – успішно змінюється режим освітлення з простого на переливання між двома кольорами та навпаки.

– Виконання команд введених за допомогою тексту або голосу – команди успішно виконуються та за потреби озвучується текстова відповідь.

– Створення/зміна/видалення сценаріїв автоматизації – успішно створюються, редагуються та видаляються сценарії автоматизації за часом та за умовами.

Система пройшла всі тест-кейси без критичних помилок. Функціональність відповідає вимогам, передбаченим у постановці задач. Голосове та текстове або ручне керування, зчитування даних з OpenWeatherMap, сценарії автоматизації, обмін даними через MQTT і збереження історії у Firebase працюють стабільно.

Розроблена інтелектуальна система голосового керування пристроями розумного дому успішно реалізована та протестована. Вона забезпечує гнучке, надійне та зручне керування освітленням та моніторинг середовища через мобільний застосунок. Результати тестування свідчать про відповідність системи заявленим вимогам, її ефективність та потенціал для подальшого вдосконалення.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Інтелектуальна система голосового керування пристроями розумного будинку є сучасним рішенням, що відповідає зростаючому попиту на автоматизацію побуту. Такий тип виробів знаходить застосування як у приватних будинках, так і в офісах, квартирах та інших приміщеннях, де необхідне зручне, бездротове та інтуїтивно зрозуміле керування освітленням і моніторинг довкілля.

Ринок розумних систем стрімко розвивається в Україні, а наявність альтернатив у вигляді готових комерційних систем (наприклад, Tuya, Xiaomi Smart Home) не перекриває попит на кастомізовані, відкриті та адаптивні системи. Запропоноване рішення є модульним, бюджетним і гнучким у налаштуванні, що створює конкурентну перевагу у сфері індивідуальних і малих рішень.

Основними конкурентами є готові закриті екосистеми розумного дому. Їхні недоліки полягають у високій вартості, обмежених можливостях адаптації та відсутності відкритого API. Цей продукт, на відміну від них, базується на відкритих технологіях (MQTT, Firebase, Android-застосунок), що дозволяє гнучко адаптувати його до потреб конкретного користувача.

Очікувані споживачі – ентузіасти IoT, фахівці в галузі автоматизації, розробники систем «розумного» дому, а також приватні користувачі, які бажають впровадити недороге розумне керування у своєму домі.

4.2 Розрахункова частина

Витрати на обладнання складають (табл. 4.1).

Таблиця 4.1 – Розрахунок витрат на обладнання

№	Найменування компонента	Кількість	Ціна за одиницю, грн	Сума, грн
1	Wi-Fi модуль ESP32 DevKit V1	1	295	295
2	Wi-Fi модуль ESP8266 ESP-01 v2	2	65	130
3	DHT11	1	132	132
4	MQ-9	1	77	77
5	Блок живлення 12V/5A	1	368	368
6	GL5528	1	20	20
7	Гніздо живлення DC	1	12	12
8	Транзистор IRLZ44N	3	20	60
9	LED стрічка 5050 RGB	1 (5 м)	250	250
10	Стабілізатор LM7805	2	10	20
11	Батарейки AAA (4 шт у блістері)	2	39	78
12	Бокси для батарейок	3	17	51
Разом				1493 грн

Розріхунок витрат на програмні сервіси міститься в таблиці 4.2.

Таблиця 4.2 – Розрахунок витрат на програмні сервіси

Сервіс	Тип витрат	Орієнтовна вартість (за місяць)
ChatGPT API	використання	5 $ (~200 грн)
Google Cloud + Firebase	використання	~3 $ (~120 грн)
Разом		320грн

У реалізації проєкту взяли участь спеціалісти, оплата праці яких складає:

– інженер-розробник апаратної частини – 3 місяці, з/п 25000 грн/міс (загалом 75000 грн);

– інженер програмного забезпечення (Android) – 3 місяці, з/п 25000 грн/міс (загалом 75000 грн);

– менеджер проєкту – 3 місяці, з/п 22000 грн/міс (загалом 66000 грн);

– тестувальник (фріланс) – 2 тижні, з/п 7000 грн.

Разом на зарплату: 223000 грн.

ЄСВ (22% від зарплати): 49060 грн.

Всього з урахуванням ЄСВ: 272060 грн.

Сумарна потужність усієї апаратної частини становить орієнтовно 65 Вт, що дорівнює 0.065 кВт. Під час тестування обладнання працювало приблизно по 8 годин на день протягом 30 днів (сумарно 15 днів роботи під час етапу розробки та 15 днів роботи під час етапу тестування), тобто 240 годин загалом. Тариф на електроенергію для фізичних осіб на момент розрахунку становить 6 грн/кВт·год.

Обчислення середніх витрат на електроенергію (Сел) проводилось за формулою 4.1.

$$Сел=P \cdot T \cdot СкВт \quad (4.1)$$

де:

P = 0.065 [кВт];

E = 240 [год];

СкВт = 6 [грн/кВт·год];

Сел = 0.065 × 240 × 6 = 93.6 [грн].

Підсумок: витрати на електроенергію під час роботи системи становлять 94 грн (з урахуванням округлення).

Крім цього, протягом усього періоду розробки тривалістю 3 місяці, три інженери працювали за ноутбуками по 8 годин на день, а тестувальник — 14 днів. Середнє споживання одного ноутбука складає приблизно 60 Вт (0.06 кВт).

Інженери (2 особи):

– загальний обсяг роботи = 2 × 30 × 8 × 3 = 1440 [годин роботи];

– середнє споживання одного ноутбука = 0.06 [кВт];

– середні витрати на елелектроенергію = 0.06 × 1440 × 6 = 518.4 [грн].

Тестувальник:

– загальний обсяг роботи = 14 × 8 = 112 [годин роботи];

– середнє споживання одного ноутбука = 0.06 [кВт];

– середні витрати на елелектроенергію = 0.06 × 112 × 6 = 40.32 [грн].

Загальні витрати на електроенергію враховуючи всі вище подані розрахунки: 94 + 518 + 40 = 652 грн.

Орієнтовно розраховано наступні витрати:

– канцелярія (папір, картридж): 200 [грн];

– зв'язок та інтернет: 500 [грн].

Разом інші витрати: 700 [грн].

Загальна сума витрат на розробку (табл. 4.3).

Таблиця 4.3 – Розрахунок загальних витрат на розробку

Категорія	Сума, грн
Обладнання	1493
ПЗ/підписки	320
Зарплата з ЄСВ	272060
Електроенергія	652
Інші витрати	700
Разом	275225

Також для покриття одноразових витрат на розробку було розраховано собівартість однієї такої системи:

– витрати на обладнання для одної системи – 1493 [грн];

– одноразові витрати – 275225 [грн];

– припустимо, потрібно вийти хоча б на 70 реалізованих систем, розподілена розробка: 275225 / 70 ≈ 3932 [грн].

Відповідно собівартість однієї системи = 1493 + 3932 = 5425 грн. Завдяки цьому було розраховано ціну яку клієнт повинен буде заплатити купивши одну

таку систему та скільки систем потрібно буде продати щоб повністю покрити витрати на розробку.

Впровадження системи дозволяє економити час працівників при керуванні освітленням, моніторингу клімату, та усуває потребу в ручному контролі. Припустимо, раніше на контроль і налаштування витрачалося щонайменше 30 хв/день (0.5 год/день), середня вартість робочої години – 120 грн.

Розрахунок економічного ефекту (Е) відбувався за формулою 4.2.

$$E=(Tручн - Tавт) \times Pгод \times N \quad (4.2)$$

де:

Тручн = 0.5 [год/день];

Тавт = 0.0013 [год/день];

Ргод = 120[грн/год];

N = 365 [кількість повторень задачі за рік];

Е = (0.5 – 0.0013) × 120 * 365= 21843 [грн/рік].

У разі використання в офісних або комерційних приміщеннях, де автоматизація забезпечує ще й зменшення витрат на електроенергію, людські помилки та підвищення безпеки — цей ефект може сягати до 50000 грн/рік. Розрахунок терміну (Ток) окупності відбувався відповідно до формули 4.3.

$$Tок = E / Cроз \quad (4.3)$$

де:

Ток – термін окупності (у роках або місяцях);

Сроз – загальні витрати на розробку проєкту (грн);

Е – економічний ефект або прибуток за рік (грн/рік).

Для інвесторів, які вклали свої кошти у розробку:

– для мінімального ефекту (21843 грн/рік): Ток = 275225 / 21843 ≈ 12.6 років;

– для потенційного ефекту (50000 грн/рік): Ток = 275225 / 50000 ≈ 5.5 років.

Для клієнтів, які будуть купувати системи було замінено Сроз на суму яку платить клієнт при покупці однієї системи (5425 грн) та вартість підписки на сервіси ChatGPT API та Google Cloud (320грн) – 5425 + 320 = 5745грн:

– для мінімального ефекту (21843 грн/рік): Ток = 5745 / 21843 ≈ 0.26 років ≈ 3.2 місяців;

– для потенційного ефекту (50000 грн/рік): Ток = 5745 / 50000 ≈ 0.12 років ≈ 1.4 місяців.

У разі комерційного застосування проєкт стає рентабельним у середньостроковій перспективі, а його масштабованість дозволяє значно зменшити витрати при багаторазовому впровадженні.

4.3 Обґрунтування необхідності розробки

Автоматизовані системи керування освітленням і моніторингом мікроклімату є одними з найбільш актуальних та затребуваних рішень у сфері сучасного домогосподарства. Запропонована система повністю відповідає цим тенденціям, забезпечуючи не лише базові функції управління, а й розширений функціонал, включаючи зручний інтерфейс, голосове керування, спілкування із вбудованим ШІ-помічником, а також можливість гнучкого планування дій та автоматизації на основі умов і сценаріїв. Такі можливості рідко зустрічаються у комерційних серійних системах керування пристроями «розумного» дому, особливо у бюджетному сегменті.

Окрему увагу варто звернути на соціальну цінність системи. Завдяки реалізації дистанційного та голосового керування, а також можливості реалізації повної автоматизації, цей пристрій може значно спростити повсякденні побутові дії для людей з інвалідністю. Відсутність потреби у фізичному контакті з елементами управління підвищує рівень комфорту і автономності для користувачів із обмеженими можливостями.

Застосування недорогих, доступних компонентів, відкритих протоколів передачі даних та реалізація виконання команд без постійного підключення до

комерційних серверів забезпечує низькі експлуатаційні витрати і стабільність роботи. Це робить систему не лише технічно доцільною, але й економічно вигідною у межах обраної ніші. Практичні результати, представлені на науково-практичній конференції в рамках Днів Науки 2025, також підтвердили ефективність обраної архітектури та її потенціал для подальшої адаптації [16].

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведено комплексне дослідження існуючих систем голосового керування розумним будинком, таких як Google Home, Amazon Alexa та Home Assistant. На основі аналізу їхніх сильних і слабких сторін було сформульовано вимоги до власної системи, яка мала забезпечити гнучкість, масштабованість, автономність і підтримку голосового інтерфейсу.

Було спроєктовано архітектуру інтелектуальної системи керування пристроями розумного дому, яка передбачала використання апаратних засобів на базі ESP32 та ESP8266-01, протоколу MQTT для обміну повідомленнями, хмарної бази даних Firebase для зберігання історії подій, а також мобільного застосунку, створеного з використанням Flutter і Dart.

У рамках реалізації було розроблено як апаратну, так і програмну частину системи. Було виконано монтаж усіх компонентів, реалізовано прошивку мікроконтролерів, створено графічний інтерфейс мобільного застосунку, реалізовано голосове керування та автоматизацію дій на основі часу й умов.

Після завершення розробки було проведено повне тестування системи, яке включало методи «білої скриньки» для перевірки логіки роботи мікроконтролерів, «чорної скриньки» для оцінки функціональності мобільного застосунку, а також функціональне тестування відповідності системи поставленим завданням. Усі тест-кейси були успішно пройдені, що підтвердило стабільність і працездатність системи.

У результаті виконаної роботи було створено інтелектуальну систему голосового керування пристроями розумного будинку, яка забезпечує зручне та інтуїтивне керування освітленням і моніторинг навколишнього середовища. Система має модульну архітектуру, підтримує масштабування та інтеграцію з сервісами штучного інтелекту, що дає можливість її подальшого розвитку та комерційного застосування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Вікіпедія Інтернет речей. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Інтернет_речей (дата звернення 02.04.2025).
2. Вікіпедія Google Home. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Google_Home (дата звернення 02.04.2025).
3. Вікіпедія Amazon Alexa. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Amazon_Alexa (дата звернення 02.04.2025).
4. Вікіпедія Home Assistant. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Home_Assistant (дата звернення 02.04.2025).
5. Arduino.ua Wi-Fi модуль DevKit V1 з ESP-32 (30 pin). Веб-сайт. URL: https://arduino.ua/prod3990-wi-fi-modyl-devkit-v1-s-esp-32 (дата звернення 03.04.2025).
6. Mini-Tech Wi-Fi модуль ESP8266 ESP-01 v2. Веб-сайт. URL: https://www.mini-tech.com.ua/ua/esp8266-esp-01-wifi-modul (дата звернення 03.04.2025).
7. LED світ світлодіодна LED стрічка 12v. Веб-сайт. URL: https://led-svit.com.ua/svitlodiodna-led-strichka-12v-5050-30led-m-ip20-rgb/ (дата звернення 03.04.2025).
8. Інтернет-магазин Prom.ua блок живлення 12В5А. Веб-сайт. URL: https://prom.ua/ua/p1898382697-blok-pitaniya-12v5a.html (дата звернення 03.04.2025).
9. Arduino.ua стабілізатор напруги LM7805. Веб-сайт. URL: https://arduino.ua/prod1844-stabilizator-napryajeniya-lm7805-to-220?srsltid=AfmBOopQFMKh1KQ7HxsQDpQbAfXGIyvstG6PYdzdZYsbUkhUgI7wZgmx (дата звернення 03.04.2025).
10. Mini-Tech IRLZ44N. Веб-сайт. URL: https://www.mini-tech.com.ua/ua/polevoy-n-kanalniy-tranzistor-irlz44n (дата звернення 03.04.2025).
11. Arduino.ua модуль датчика вологості та температури DHT11. Веб-сайт. URL: https://arduino.ua/prod602-modyl-datchika-vlajnosti-i-temperatyri-

dht11?srsltid=AfmBOoooPPInr0fmheKryxdifDv0PpFhKhnhhVMMwWu-ZFNOV5FOLp6d (дата звернення 04.04.2025).

12. Arduino.ua модуль датчика газу MQ-9. Веб-сайт. URL: https://arduino.ua/prod1238-modyl-datchika-gaza-mq-9?srsltid=AfmBOopU6mh1lm_oSQSemFOHwOMwGiRKrztxKCJkFa4AX2tiOwfWZnhu (дата звернення 04.04.2025).

13. Mini-Tech модуль GL5528. Веб-сайт. URL: https://www.mini-tech.com.ua/ua/modul-datchik-sveta-na-osnove-fotorezistora (дата звернення 04.04.2025).

14. Епіцентр Батарейка HausMark Basic Power AAA 4 шт. Веб-сайт. URL: https://epicentrk.ua/ua/shop/batareyka-hausmark-basic-power-aaa-4-sht-mst-al4aaa.html (дата звернення 09.04.2025).

15. Rozetka Акумуляторний/батарейний відсік на 2 x AAA. Веб-сайт. URL: https://rozetka.com.ua/ua/477461119/p477461119/ (дата звернення 09.04.2025).

16. Береза О. ІНТЕЛЕКТУАЛЬНА СИСТЕМА ГОЛОСОВОГО КЕРУВАННЯ ПРИСТРОЯМИ “РОЗУМНОГО” ДОМУ // Збірник тез за матеріалами студентської науково-практичної конференції в рамках Днів Науки 2025 (секція «Комп’ютерних технологій»). – Тернопіль: Галицький фаховий коледж ім. В’ячеслава Чорновола, 2025. – С. 6–10 (дата звернення).

ДОДАТКИ

Додаток А

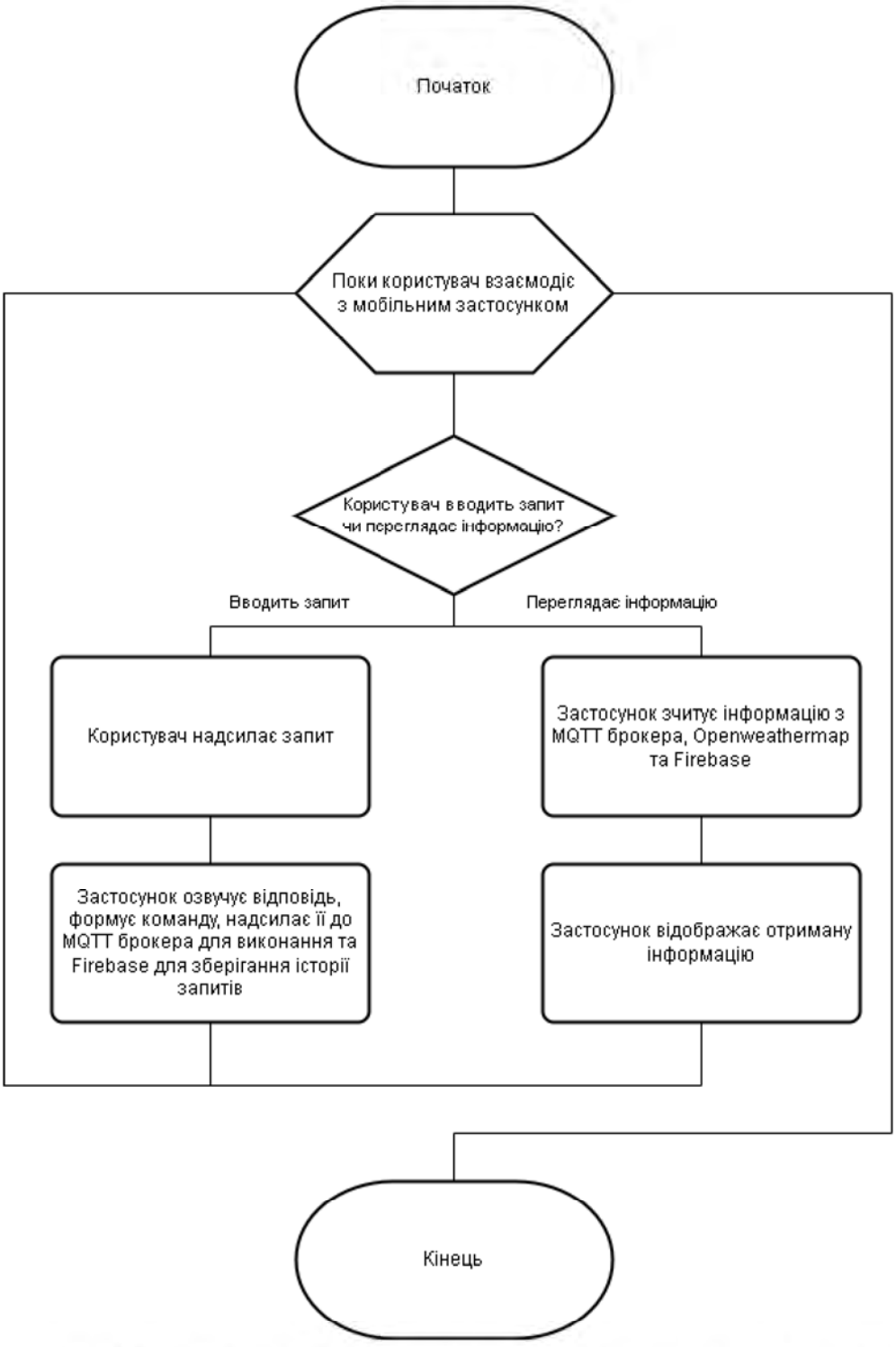

Рисунок 1 – Блок-схема алгоритму взаємодії користувача і мобільного застосунку

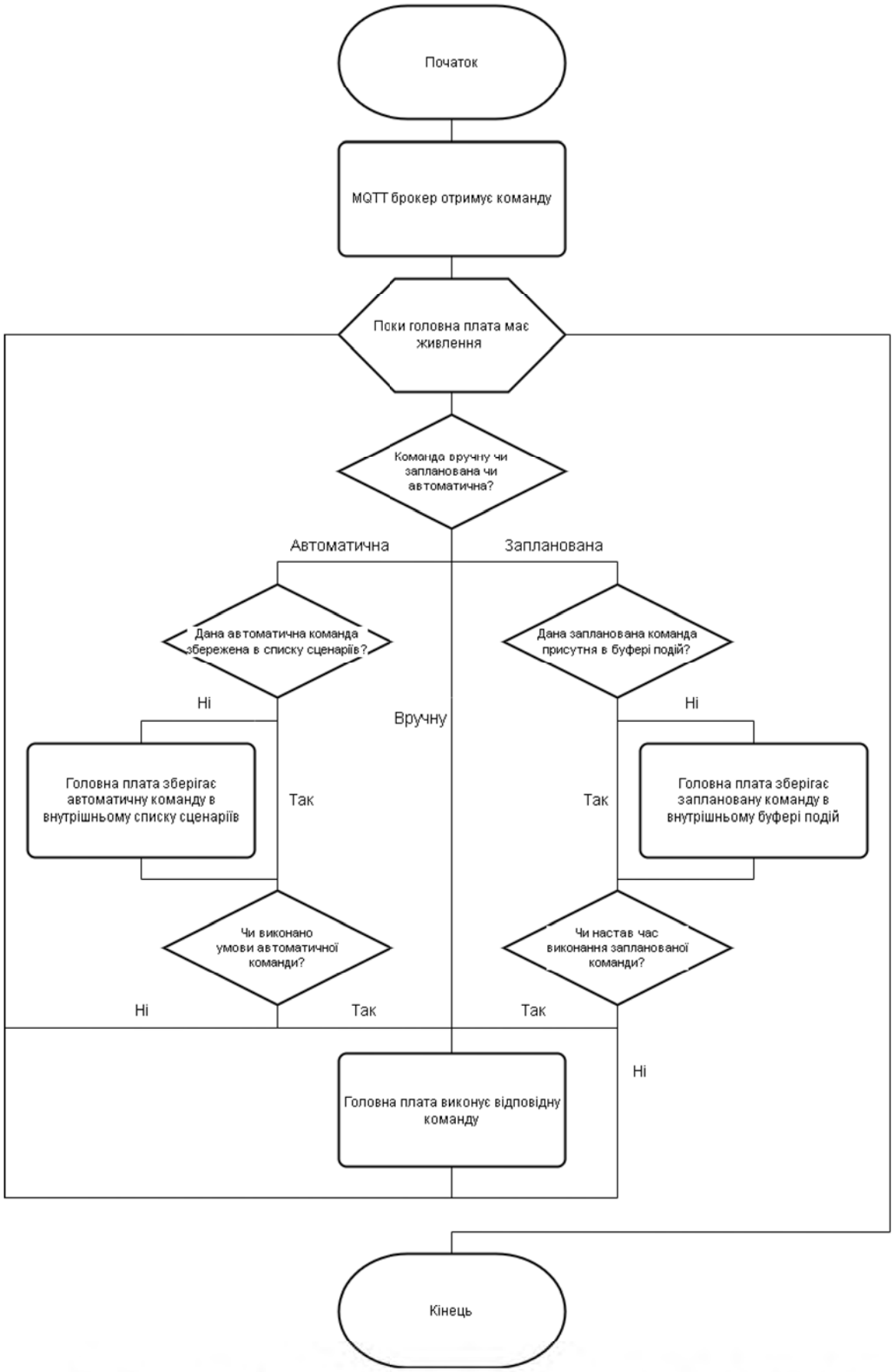

Рисунок 2 – Блок-схема алгоритму взаємодії головної плати та MQTT брокера

Рисунок 3 – Блок-схема алгоритму взаємодії сенсорних модулів та MQTT брокера

Додаток Б

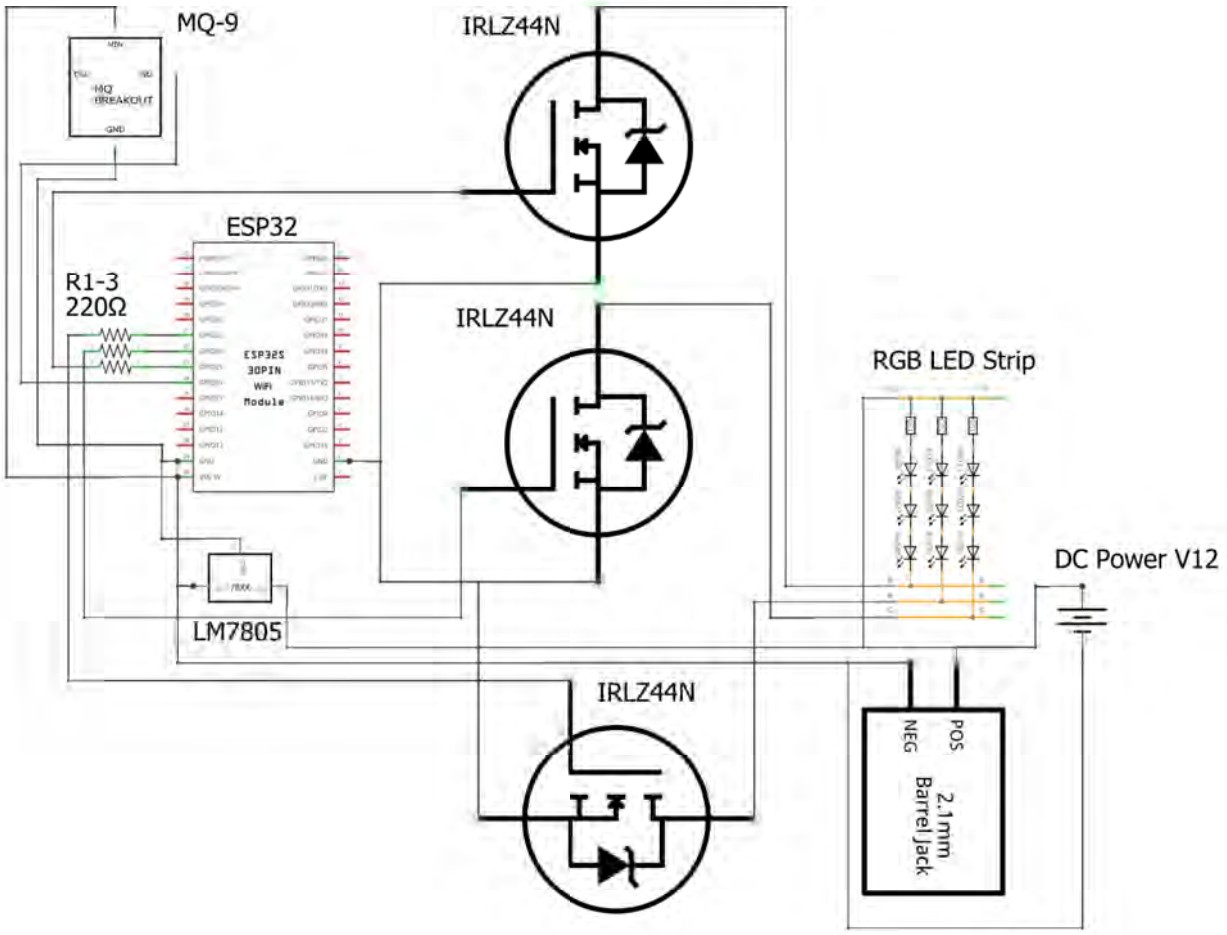

Рисунок 1 – Принципово-електрична схема головного модуля

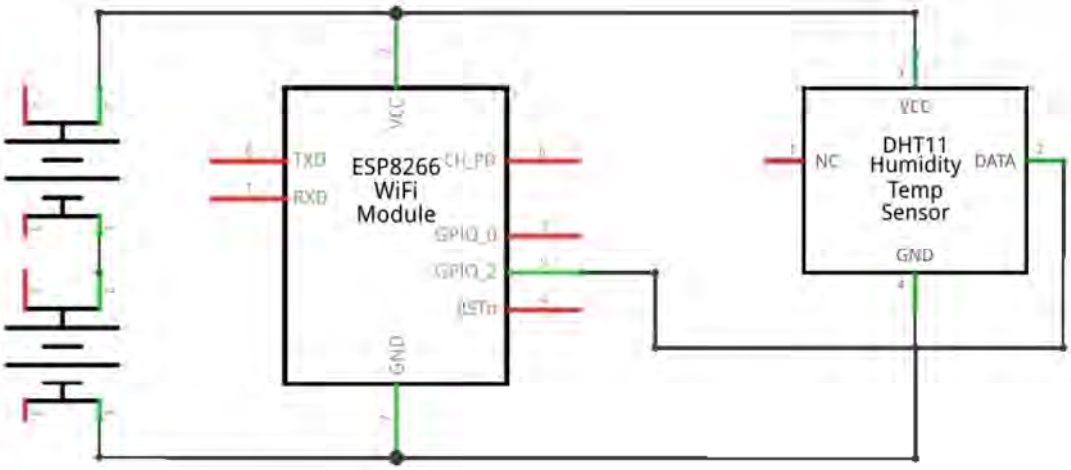

Рисунок 2 – Принципово-електрична схема сенсорного модуля з DHT11

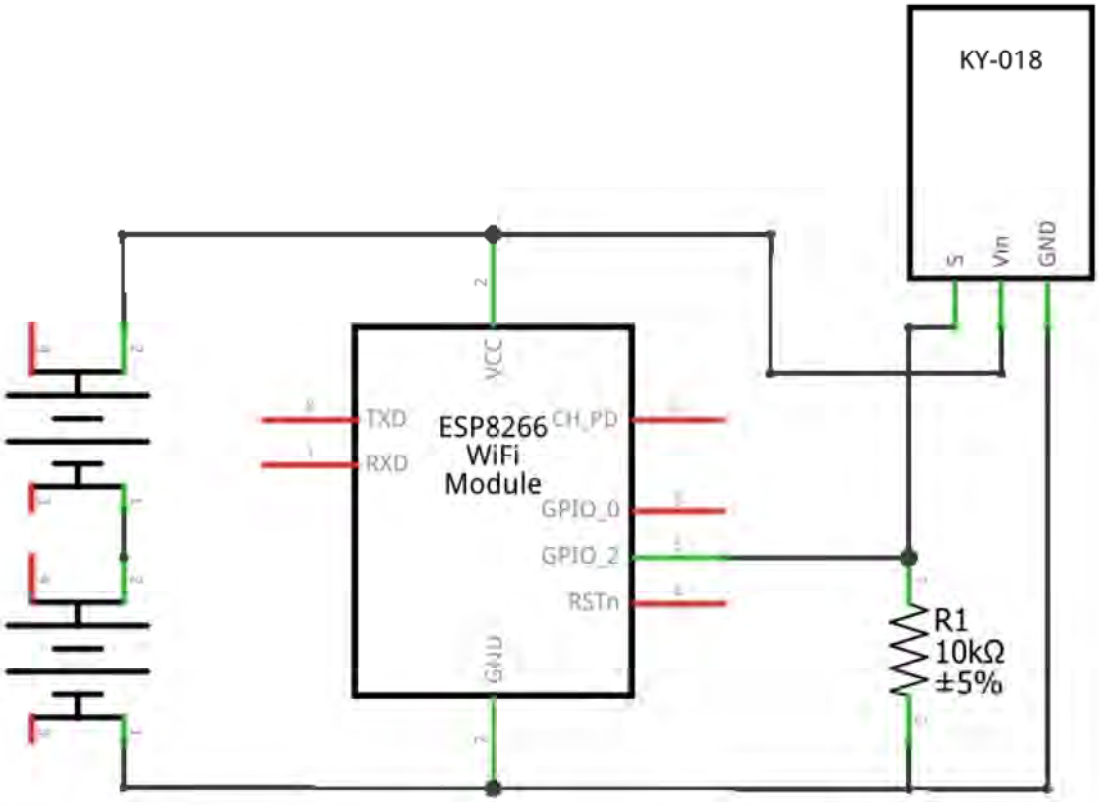

Рисунок 3 – Принципово-електрична схема сенсорного модуля з GL5528

Додаток В

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

// Wi-Fi налаштування
// const char* ssid = "MAIN";
// const char* password = "0982655181andriy";
const char* ssid = "iot_21_bereza";
const char* password = "smarthome";

// WiFiUDP ntpUDP;

const char* automationTopic = "Automation";
JsonDocument automationDoc;
unsigned long lastAutomationCheck = 0;

int mq9Value = 0;
int lightValue = 0;
float temperature = 0.0;
int humidity = 0;
bool autoLightMode = false;

// MQTT налаштування
// const char* mqtt_server = "192.168.0.106"; // IP адреса брокера
const char* mqtt_server = "192.168.137.1"; // IP адреса брокера
const char* mqtt_user = "esp32";
const char* mqtt_password = "12345";

// Налаштування GPIO
#define MQ9_PIN 34
#define RED_PIN 32
#define GREEN_PIN 33
#define BLUE_PIN 25

WiFiClient espClient;
PubSubClient client(espClient);

// Поточні кольори
int currentRed = 50, currentGreen = 50, currentBlue = 50;
int initialRed = 50, initialGreen = 50, initialBlue = 50;

// Змінні для режиму переливання
bool isPhaseActive = false;
int phaseR1 = 255, phaseG1 = 0, phaseB1 = 0; // Колір 1
int phaseR2 = 0, phaseG2 = 0, phaseB2 = 255; // Колір 2
int phaseDuration = 5000; // Час переходу в мс (за замовчуванням 5
секунд)
unsigned long lastTransitionTime = 0;
bool transitionToSecondColor = true; // Вказує, до якого кольору
зараз перехід
```

```
bool isPulsing = false;
int pulseRed = 0, pulseGreen = 0, pulseBlue = 0; // Колір
пульсації
int pulseIntensity = 50; // Відсоток інтенсивності пульсації (від
0 до 100)
int pulseDuration = 5; //seconds
int pulseSpeed = 500; //milliseconds

void setupPWM() {
  ledcSetup(0, 5000, 8); // Канал 0, частота 5 кГц, розрядність 8
біт
  ledcSetup(1, 5000, 8); // Канал 1
  ledcSetup(2, 5000, 8); // Канал 2
  ledcAttachPin(RED_PIN, 0);
  ledcAttachPin(GREEN_PIN, 1);
  ledcAttachPin(BLUE_PIN, 2);
}

void setColor(int red, int green, int blue) {
  currentRed = red;
  currentGreen = green;
  currentBlue = blue;
  ledcWrite(0, currentRed);
  ledcWrite(1, currentGreen);
  ledcWrite(2, currentBlue);
}

void smoothTransition(int targetRed, int targetGreen, int
targetBlue, int transitionTime) {
  int stepDelay = 10; // Час між кроками (мс)
  int steps = transitionTime / stepDelay;

  for (int i = 0; i <= steps; i++) {
    int newRed = currentRed + ((targetRed - currentRed) * i /
steps);
    int newGreen = currentGreen + ((targetGreen - currentGreen) *
i / steps);
    int newBlue = currentBlue + ((targetBlue - currentBlue) * i /
steps);

    ledcWrite(0, newRed);
    ledcWrite(1, newGreen);
    ledcWrite(2, newBlue);
    delay(stepDelay);
  }

  currentRed = targetRed;
  currentGreen = targetGreen;
  currentBlue = targetBlue;
}
```

```
void setup_wifi() {
  delay(10);
  Serial.println("Підключення до Wi-Fi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWi-Fi підключено");
  Serial.println(WiFi.localIP());
}

// Обробка отриманого повідомлення
void callback(char* topic, byte* payload, unsigned int length) {
  String data;
  for (unsigned int i = 0; i < length; i++) {
    data += (char)payload[i];
  }
  data.trim(); // Видаляємо зайві пробіли

  if (String(topic) ==
"BerezaOleksandr/berezaLight/lightData/colors") {
    // Розділяємо отримане повідомлення за пробілами
    int targetRed, targetGreen, targetBlue;
    sscanf(data.c_str(), "%d %d %d", &targetRed, &targetGreen,
&targetBlue);
    targetRed = constrain(targetRed, 0, 255);
    targetGreen = constrain(targetGreen, 0, 255);
    targetBlue = constrain(targetBlue, 0, 255);

    // Плавний перехід до нового кольору
    if (!isPhaseActive) {
      Serial.println("Плавний перехід до нового кольору:" +
String(targetRed) + String(targetGreen) + String(targetBlue));
      smoothTransition(targetRed, targetGreen, targetBlue, 1000);
// 1 секунда
    }
  }

  if (String(topic) ==
"BerezaOleksandr/berezaLight/lightData/isPhaseActive") {
    isPhaseActive = (data == "false");
    isPhaseActive = (data == "true");
    Serial.println(isPhaseActive ? "Режим переливання увімкнено" :
"Режим переливання вимкнено");
  }

  if (String(topic) == "Devices/ESP01_DHT") {
  String payloadStr = data;
  int separatorIndex = payloadStr.indexOf(';');

  if (separatorIndex != -1) {
```

```
    String tempStr = payloadStr.substring(0, separatorIndex);
    String humStr = payloadStr.substring(separatorIndex + 1);

    temperature = tempStr.toFloat();
    humidity = humStr.toInt();

    Serial.println("Дані температури та вологості оновлено.");
    Serial.print("🌡 Температура: ");
    Serial.println(temperature);
    Serial.print("💧 Вологість: ");
    Serial.println(humidity);
  }
}
  if(String(topic) == "Devices/D1_GL5528"){
    String payloadStr = data;
    lightValue = payloadStr.toInt();
    Serial.println("Дані освітлення навулиці оновлено.");
  }

  if (String(topic) ==
"BerezaOleksandr/berezaLight/lightData/phaseColors") {
    // Розділяємо отримане повідомлення

    sscanf(data.c_str(), "%d %d %d %d %d %d %d",
           &phaseR1, &phaseG1, &phaseB1,
           &phaseR2, &phaseG2, &phaseB2,
           &phaseDuration);
    phaseDuration *= 1000; // Перетворення секунд у мс
    Serial.println("Дані режиму переливання оновлено.");
  }

  if (String(topic) ==
"BerezaOleksandr/berezaLight/lightData/ChatGPTAnswerPulseColors")
{
    sscanf(data.c_str(), "%d %d %d", &pulseRed, &pulseGreen,
&pulseBlue);
    pulseRed = constrain(pulseRed, 0, 255);
    pulseGreen = constrain(pulseGreen, 0, 255);
    pulseBlue = constrain(pulseBlue, 0, 255);
    Serial.println("Дані пульсації кольору оновлено.");
  }

  if (String(topic) ==
"BerezaOleksandr/berezaLight/lightData/ChatGPTAnswerPulse") {
    isPulsing = (data == "false");
    isPulsing = (data == "true");
    Serial.println(isPulsing ? "Режим пульсації увімкнено" :
"Режим пульсації вимкнено");
  }

  if (String(topic) == "Automation") {
```

```
    deserializeJson(automationDoc, data);
    Serial.println("Дані автоматизації оновлено.");
  }

  if (String(topic) ==
"BerezaOleksandr/berezaLight/lightData/autoLightMode") {
  autoLightMode = (data == "true");
  Serial.println(autoLightMode ? "🔆 Автоматичне освітлення:
Увімкнено" : "🌙 Автоматичне освітлення: Вимкнено");
  }
}

void handleAutoLight() {
  if (!autoLightMode || isPhaseActive) return;

  // Нормалізація значення освітлення (наприклад 0-1023 → 0-1)
  float brightnessFactor = constrain(lightValue / 800.0, 0.0,
1.0);

  // Обчислення кольорів
  int autoRed = currentRed * brightnessFactor;
  int autoGreen = currentGreen * brightnessFactor;
  int autoBlue = currentBlue * brightnessFactor;

  // Виведення
  ledcWrite(0, autoRed);
  ledcWrite(1, autoGreen);
  ledcWrite(2, autoBlue);

  Serial.print("◑ AutoLight → Яскравість: ");
  Serial.print(brightnessFactor, 2);
  Serial.print(" → RGB: ");
  Serial.print(autoRed); Serial.print(", ");
  Serial.print(autoGreen); Serial.print(", ");
  Serial.println(autoBlue);
}

void handleAutomations() {
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("✖ Не вдалося отримати локальний час");
    return;
  }

  char currentTimeStr[20];
  sprintf(currentTimeStr, "%02d.%02d.%04d %02d:%02d",
          timeinfo.tm_mday, timeinfo.tm_mon + 1, timeinfo.tm_year
+ 1900,
          timeinfo.tm_hour, timeinfo.tm_min);

  JsonArray automations = automationDoc["automations"];
```

```
  for (int i = 0; i < automations.size(); i++) {
    JsonObject automation = automations[i];
    String type = automation["type"];
    String command = automation["command"];
    String args = automation["args"];
    String datetime = automation["datetime"];
    bool repeat = automation["repeat"];

    datetime.trim();
    String currentTimeFormatted = String(currentTimeStr);
    currentTimeFormatted.trim();

    if (type == "time" && datetime == currentTimeFormatted) {
      Serial.println("🕒 Виконання автоматизації за часом");
      executeCommand(command, args);

      if (!repeat) {
        automations.remove(i);
        i--;
      } else {
        struct tm parsedTime = {};
        if (strptime(datetime.c_str(), "%d.%m.%Y %H:%M",
&parsedTime)) {
          parsedTime.tm_sec = 0;      // обов'язково!
          parsedTime.tm_isdst = -1;   // DST — хай система сама
вирішує

          time_t t = mktime(&parsedTime);
          if (t == -1) {
            Serial.println("❌ mktime повернув -1, не вдалося
створити час");
            return;
          }

          t += 86400; // +1 день
          struct tm* newTime = localtime(&t);

          char buffer[20];
          sprintf(buffer, "%02d.%02d.%04d %02d:%02d",
                  newTime->tm_mday, newTime->tm_mon + 1, newTime-
>tm_year + 1900,
                  newTime->tm_hour, newTime->tm_min);

          automation["datetime"] = String(buffer);

          // 📨 Надсилання назад в MQTT
          String updatedJson;
          serializeJson(automationDoc, updatedJson);
          client.publish("Automation", updatedJson.c_str(), true);
          Serial.println("✅ Оновлена автоматизація надіслана");
        } else {
```

```
          Serial.println("⚠ Не вдалося розпарсити дату для
повтору");
        }
      }
    }

    //Автоматизація по умові
    if (type == "condition") {
      String parameter = automation["parameter"];
      String operatorType = automation["operator"];
      float value = automation["value"];

      float sensorValue = 0;
      if (parameter == "temperature") sensorValue = temperature;
      else if (parameter == "humidity") sensorValue = humidity;
      else if (parameter == "light") sensorValue = lightValue;
      else if (parameter == "mq9") sensorValue = mq9Value;

      bool conditionMet = false;
      if (operatorType == ">") conditionMet = sensorValue > value;
      else if (operatorType == "<") conditionMet = sensorValue <
value;
      else if (operatorType == ">=") conditionMet = sensorValue >=
value;
      else if (operatorType == "<=") conditionMet = sensorValue <=
value;
      else if (operatorType == "==") conditionMet = sensorValue ==
value;
      else if (operatorType == "!=") conditionMet = sensorValue !=
value;

      if (conditionMet) {
        Serial.println("✅ Виконання автоматизації за умовою");
        executeCommand(command, args);
      }
    }
  }
}

void executeCommand(String command, String args) {
  if (command == "light") {
    int r, g, b;
    sscanf(args.c_str(), "%d %d %d", &r, &g, &b);
    smoothTransition(r, g, b, 1000);
    Serial.println("Світло оновлено через автоматизації.");
  }
  else if (command == "lightPhase") {
    sscanf(args.c_str(), "%d %d %d %d %d %d %d",
           &phaseR1, &phaseG1, &phaseB1,
           &phaseR2, &phaseG2, &phaseB2,
           &phaseDuration);
```

```
    phaseDuration *= 1000;
    isPhaseActive = true;
    Serial.println("Світло переливання оновлено через
автоматизації.");
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Підключення до MQTT...");
    if (client.connect("Bereza_Oleksandr_ESP32_Client")) {
      Serial.println("Підключено");
      client.subscribe("BerezaOleksandr/berezaLight/lightData/colo
rs");
      client.subscribe("BerezaOleksandr/berezaLight/lightData/isPh
aseActive");
      client.subscribe("BerezaOleksandr/berezaLight/lightData/phas
eColors");
      client.subscribe("BerezaOleksandr/berezaLight/lightData/Chat
GPTAnswerPulseColors");
      client.subscribe("BerezaOleksandr/berezaLight/lightData/Chat
GPTAnswerPulse");
      client.subscribe("BerezaOleksandr/berezaLight/lightData/auto
LightMode");
      client.subscribe("Devices/ESP01_DHT");
      client.subscribe("Devices/D1_GL5528");
      client.subscribe("Automation");
    } else {
      Serial.println("Не вдалось підключитись, повтор через 5
секунд");
      delay(5000);
    }
  }
}

void handlePhaseMode() {
  if (isPhaseActive) {
    unsigned long currentTime = millis();
    if (currentTime - lastTransitionTime >= phaseDuration) {
      lastTransitionTime = currentTime;

      if (transitionToSecondColor) {
        smoothTransition(phaseR2, phaseG2, phaseB2,
phaseDuration);
      } else {
        smoothTransition(phaseR1, phaseG1, phaseB1,
phaseDuration);
      }

      transitionToSecondColor = !transitionToSecondColor;
    }
  }
```

```
}

void pulseColors() {
    unsigned long startTime = millis(); // Початковий час
    initialRed = currentRed;
    initialGreen = currentGreen;
    initialBlue = currentBlue;
    while (millis() - startTime < (pulseDuration * 1000) ) { //
Пульсувати n секунд
        // Колір пульсації з додаванням до поточного кольору
        int addedRed = currentRed + (pulseRed - currentRed) *
pulseIntensity / 100;
        int addedGreen = currentGreen + (pulseGreen -
currentGreen) * pulseIntensity / 100;
        int addedBlue = currentBlue + (pulseBlue - currentBlue) *
pulseIntensity / 100;

        // Плавний перехід до змішаного кольору
        smoothTransition(addedRed, addedGreen, addedBlue,
pulseSpeed);

        // Повернення до поточного кольору
        smoothTransition(initialRed, initialGreen, initialBlue,
pulseSpeed);
    }

    // Надіслати повідомлення з флагом retain
    client.publish("BerezaOleksandr/berezaLight/lightData/ChatGPTA
nswerPulse", "false", true);
}

void readAndPublishMQ9() {
  int mq9Value = analogRead(MQ9_PIN); // Зчитування з датчика
  char msg[10];
  sprintf(msg, "%d", mq9Value);
  client.publish("Devices/mq9", msg, true);
}

void setup() {
  Serial.begin(115200);
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
  pinMode(MQ9_PIN, INPUT);
  setupPWM();

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

  // □ Синхронізація часу через NTP
```

```
  configTime(10800, 0, "pool.ntp.org", "time.nist.gov"); // UTC+3
(Київ)

  Serial.print("□ Очікування синхронізації часу");
  while (!time(nullptr)) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("\n✓ Час синхронізовано");
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  // Обробка режиму переливання
  handlePhaseMode();


  if (isPulsing) {
        pulseColors();
        isPulsing = false; // Виконати пульсацію тільки один раз
після отримання true
        smoothTransition(initialRed, initialGreen, initialBlue,
pulseSpeed);
    }
  static unsigned long lastMQ9Read = 0;
  unsigned long now = millis();
  if (now - lastMQ9Read > 5000) { // кожні 5 секунд
    readAndPublishMQ9();
    lastMQ9Read = now;
  }

  if (millis() - lastAutomationCheck > 5000) { // кожні 5 сек
    handleAutomations();
    lastAutomationCheck = millis();
    handleAutoLight();
  }
}
```

Додаток Г

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "MAIN";
const char* password = "0982655181andriy";
// const char* mqtt_server = "192.168.137.1";
const char* mqtt_server = "192.168.0.106";
const char* mqtt_user = "esp_d1_gl5528";
const char* mqtt_password = "12345";

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(500);
}

void reconnect() {
  while (!client.connected()) {
    client.connect("ESP_D1_GL5528");
  }
}

void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  int lightLevel = analogRead(A0);  // читає значення 0-1023
  String payload = String(lightLevel);
  client.publish("Devices/D1_GL5528", payload.c_str());

  delay(2000);  // кожні 5 секунд
}
```

Додаток Д

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

#define DHTPIN 2  // GPIO2
#define DHTTYPE DHT11

const char* ssid = "MAIN";
const char* password = "0982655181andriy";
const char* mqtt_server = "192.168.0.106";
const char* mqtt_user = "esp01s_dht";
const char* mqtt_password = "12345";

WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(DHTPIN, DHTTYPE);

void setup_wifi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(500);
}

void reconnect() {
  while (!client.connected()) {
    client.connect("ESP01S_DHT");
  }
}

void setup() {
  dht.begin();
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  int h = dht.readHumidity();
  float t = dht.readTemperature();

  if (!isnan(h) && !isnan(t)) {
    String payload = String(t) + "; " + String(h);
    client.publish("Devices/ESP01_DHT", payload.c_str());
  }

  delay(30000);
}
```

Додаток Ґ

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:google_speech/google_speech.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:cloud_firestore/cloud_firestore.dart' as
firestore;
import 'package:permission_handler/permission_handler.dart';
import 'package:intl/intl.dart';
import 'package:google_speech/speech_client_authenticator.dart';
import
'package:record_platform_interface/record_platform_interface.dart'
;
import 'package:path_provider/path_provider.dart';
import 'dart:async';
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:googleapis/texttospeech/v1.dart';
import 'package:googleapis_auth/googleapis_auth.dart';
import 'package:audioplayers/audioplayers.dart';
import 'package:mqtt_client/mqtt_client.dart';
import 'package:mqtt_client/mqtt_server_client.dart';
import 'package:weather_icons/weather_icons.dart';
import 'package:flutter/services.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:table_calendar/table_calendar.dart';

String recognizedText = 'Натисніть кнопку та скажіть щось для
запиту.';
String recognizedTextFull = '';
String responseForUser =
    'Задайте запит, наприклад включити світло певного кольору або
змінити режим світла. Також я можу відповідати на ваші питання та
маю доступ до поточної погоди.';
String reqSendState = '';
String userInputHintText = 'Натисніть щоб написати';
bool isPhaseActive = false;
Color color240 = const Color.fromARGB(255, 240, 240, 240);
Color purpleAccentColor = const Color.fromARGB(255, 192, 127,
253);
Color darkPurpleAccentColor = const Color.fromARGB(255, 133, 74,
189);
Color navBarColor = const Color.fromARGB(255, 20, 18, 24);

class LightScreen extends StatefulWidget {
  const LightScreen({super.key});

  @override
  State<LightScreen> createState() => _LightScreenState();
}
```

```
class _LightScreenState extends State<LightScreen> {
  String localBrokerIp = '192.168.0.106';
  String vpnBrokerIp = '192.168.0.144';
  int red = 255;
  int green = 255;
  int blue = 255;
  int red1 = 255;
  int green1 = 255;
  int blue1 = 255;
  int red2 = 50;
  int green2 = 50;
  int blue2 = 50;
  int phaseDuration = 1;
  bool isLoading = true;
  bool retain = false;
  bool isAutoLightMode = false;
  Color accentColor = purpleAccentColor;

  @override
  void initState() {
    super.initState();
    _loadSettings();
    _loadLightColor();

    SystemChrome.setSystemUIOverlayStyle(const
SystemUiOverlayStyle(
      systemNavigationBarColor:
          Color.fromARGB(255, 20, 18, 24),
      systemNavigationBarIconBrightness:
          Brightness.light,
    ));
  }

  Future<void> _loadSettings() async {
    final settings = await SettingsManager.loadSettings();
    setState(() {
      localBrokerIp = settings['localIp'];
      vpnBrokerIp = settings['vpnIp'];
      accentColor = settings['accentColor'];
    });
  }

  Future<void> sendAutoLightModeToMQTT(bool value) async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic =
'BerezaOleksandr/berezaLight/lightData/autoLightMode';
    final payload = value ? 'true' : 'false';

    await _loadSettings();
    final client = MqttServerClient(localBrokerIp, clientId);
```

```
    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      try {
        await client.connect();
        return client.connectionStatus?.state ==
MqttConnectionState.connected;
      } catch (_) {
        return false;
      }
    }

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(topic, MqttQos.atLeastOnce,
builder.payload!,
        retain: true);
    client.disconnect();
  }

  Future<void> _loadLightColor() async {
    var lightDoc = await firestore.FirebaseFirestore.instance
        .collection('MQTT')
        .doc('lightData')
        .get();

    if (lightDoc.exists) {
      var lightData = lightDoc.data()!;
      setState(() {

        isPhaseActive = lightData['isPhaseActive'] == 1;

        String phaseColors = lightData['phaseColors'] ?? '';
        List<String> colorParts = phaseColors.split(' ');

        if (colorParts.length == 7) {
          red1 = int.tryParse(colorParts[0]) ?? 0;
          green1 = int.tryParse(colorParts[1]) ?? 0;
          blue1 = int.tryParse(colorParts[2]) ?? 0;

          red2 = int.tryParse(colorParts[3]) ?? 0;
          green2 = int.tryParse(colorParts[4]) ?? 0;
          blue2 = int.tryParse(colorParts[5]) ?? 0;
```

```
          phaseDuration = int.tryParse(colorParts[6]) ?? 1;
          phaseDuration = phaseDuration.clamp(1, 120); //
Обмеження 1-120 секунд
        } else {
          print("Неправильний формат phaseColors: $phaseColors");
        }


        String colorData = lightData['colors'] ?? '255 255 255';
        List<String> singleColorParts = colorData.split(' ');

        red = int.tryParse(singleColorParts[0]) ?? 255;
        green = int.tryParse(singleColorParts[1]) ?? 255;
        blue = int.tryParse(singleColorParts[2]) ?? 255;
      });
    }
  }

  Future<void> sendLightToMQTT(int red, int green, int blue) async
{
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic = 'BerezaOleksandr/berezaLight/lightData/colors';
    final payload = '$red $green $blue';
    await _loadSettings();

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      client.logging(on: true);
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;

      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();

        if (client.connectionStatus?.state ==
MqttConnectionState.connected) {
          print('Підключено до $brokerIp!');
          return true;
        } else {
          print('Стан підключення:
${client.connectionStatus?.state}');
        }
      } catch (e) {
        print('Помилка підключення до $brokerIp: $e');
      }
      return false;
    }

    final client = MqttServerClient(localBrokerIp, clientId);
```

```
    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(
      topic,
      MqttQos.atLeastOnce,
      builder.payload!,
      retain: retain,
    );

    print('Дані успішно відправлено: $payload (Retain: $retain)');

    client.disconnect();
  }

  void _onDisconnected() {
    print('Відключено від брокера.');
  }

  void refresh() async {
    await _loadLightColor();
  }

  void _showColorPickerDialog() {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        int tempRed = red;
        int tempGreen = green;
        int tempBlue = blue;

        return StatefulBuilder(
          builder: (context, setState) {
            return AlertDialog(
              backgroundColor: const Color.fromARGB(255, 30, 30,
30),
              title: const Text(
                'Вибір кольору',
                style: TextStyle(color: Colors.white),
              ),
              content: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  Container(
                    width: 80,
```

```
                    height: 80,
                    decoration: BoxDecoration(
                      color: Color.fromARGB(255, tempRed,
tempGreen, tempBlue),
                      shape: BoxShape.circle,
                      border: Border.all(color: Colors.white30,
width: 2),
                    ),
                  ),
                  const SizedBox(height: 10),
                  _buildColorSlider("Red", tempRed, (value) {
                    setState(() => tempRed = value);
                  }),
                  _buildColorSlider("Green", tempGreen, (value) {
                    setState(() => tempGreen = value);
                  }),
                  _buildColorSlider("Blue", tempBlue, (value) {
                    setState(() => tempBlue = value);
                  }),
                  Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      const Text(
                        'Зберігати значення?',
                        style: TextStyle(color: Colors.white),
                      ),
                      IconButton(
                        icon: Icon(
                          retain
                              ? Icons.check_box
                              : Icons.check_box_outline_blank,
                          color: retain ? accentColor :
Colors.grey,
                        ),
                        onPressed: () {
                          setState(() {
                            retain = !retain;
                          });
                        },
                      ),
                    ],
                  ),
                ],
              ),
              actions: [
                TextButton(
                  onPressed: () => Navigator.pop(context),
                  child: const Text("Скасувати",
                      style: TextStyle(color: Colors.red)),
                ),
                TextButton(
                  onPressed: () async {
```

```
                    await firestore.FirebaseFirestore.instance
                        .collection('MQTT')
                        .doc('lightData')
                        .update({
                      'colors': "$tempRed $tempGreen $tempBlue",
                      'isPhaseActive': 0,
                      'time of change': firestore.Timestamp.now(),
                    });
                    await sendLightPhaseIsActiveToMQTT(false);
                    await sendLightToMQTT(tempRed, tempGreen,
tempBlue);

                    Navigator.pop(context);
                    refresh();
                  },
                  child: const Text("Зберегти",
                      style: TextStyle(color: Colors.green)),
                ),
              ],
            );
          },
        );
      },
    );
  }

  Future<void> sendLightPhaseToMQTT(
      int r1, int g1, int b1, int r2, int g2, int b2, int seconds)
async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic =
'BerezaOleksandr/berezaLight/lightData/phaseColors';
    final payload = '$r1 $g1 $b1 $r2 $g2 $b2 $seconds';
    await _loadSettings();

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      client.logging(on: true);
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;

      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();

        if (client.connectionStatus?.state ==
MqttConnectionState.connected) {
          print('Підключено до $brokerIp!');
          return true;
        } else {
```

```
          print('Стан підключення:
${client.connectionStatus?.state}');
        }
      } catch (e) {
        print('Помилка підключення до $brokerIp: $e');
      }
      return false;
    }

    final client = MqttServerClient(localBrokerIp, clientId);

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(
      topic,
      MqttQos.atLeastOnce,
      builder.payload!,
    );

    print('Дані успішно відправлено: $payload');

    client.disconnect();
  }

  Future<void> sendLightPhaseToFirebaseAndMQTT(
      int r1, int g1, int b1, int r2, int g2, int b2, int seconds)
async {
    await firestore.FirebaseFirestore.instance
        .collection('MQTT')
        .doc('lightData')
        .update({
      'phaseColors': '$r1 $g1 $b1 $r2 $g2 $b2 $seconds',
      'isPhaseActive': 1,
      'time of change': firestore.Timestamp.now(),
    });
    await sendLightPhaseToMQTT(r1, g1, b1, r2, g2, b2, seconds);
  }

  void _showLightPhaseDialog() {
    int tempRed1 = red1;
    int tempGreen1 = green1;
    int tempBlue1 = blue1;

    int tempRed2 = red2;
```

```
    int tempGreen2 = green2;
    int tempBlue2 = blue2;

    int tempPhaseDuration = phaseDuration;

    int activeColorIndex = 0;

    showDialog(
      context: context,
      builder: (BuildContext context) {
        return StatefulBuilder(
          builder: (context, setState) {
            return AlertDialog(
              backgroundColor: const Color.fromARGB(255, 30, 30,
30),
              title: const Text(
                'Налаштування переливання',
                style: TextStyle(color: Colors.white),
              ),
              content: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  // Градієнтний контейнер
                  Container(
                    width: 80,
                    height: 80,
                    decoration: BoxDecoration(
                      gradient: LinearGradient(
                        colors: [
                          Color.fromARGB(255, tempRed1,
tempGreen1, tempBlue1),
                          Color.fromARGB(255, tempRed2,
tempGreen2, tempBlue2)
                        ],
                        begin: Alignment.topLeft,
                        end: Alignment.bottomRight,
                      ),
                      shape: BoxShape.circle,
                      border: Border.all(color: Colors.white30,
width: 2),
                    ),
                  ),
                  const SizedBox(height: 10),

                  SizedBox(
                    height: 300,
                    child: PageView(
                      onPageChanged: (index) {
                        setState(() => activeColorIndex = index);
                      },
                      children: [
                        // Повзунки для першого кольору
```

```
                    Column(
                      children: [
                        _buildColorSlider("Червоний (1-й
колір)", tempRed1,
                            (value) {
                          setState(() => tempRed1 = value);
                        }),
                        _buildColorSlider("Зелений (1-й
колір)", tempGreen1,
                            (value) {
                          setState(() -> tempGreen1 - value);
                        }),
                        _buildColorSlider("Синій (1-й колір)",
tempBlue1,
                            (value) {
                          setState(() => tempBlue1 = value);
                        }),
                      ],
                    ),
                    // Повзунки для другого кольору
                    Column(
                      children: [
                        _buildColorSlider("Червоний (2-й
колір)", tempRed2,
                            (value) {
                          setState(() => tempRed2 = value);
                        }),
                        _buildColorSlider("Зелений (2-й
колір)", tempGreen2,
                            (value) {
                          setState(() => tempGreen2 = value);
                        }),
                        _buildColorSlider("Синій (2-й колір)",
tempBlue2,
                            (value) {
                          setState(() => tempBlue2 = value);
                        }),
                      ],
                    ),
                  ],
                ),
              ),

              Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  const Text(
                    "Швидкість анімації (секунди)",
                    style: TextStyle(color: Colors.white),
                  ),
                  Slider(
                    value: tempPhaseDuration.toDouble(),
```

```
                        min: 1,
                        max: 120,
                        divisions: 119,
                        label: "$tempPhaseDuration сек",
                        activeColor: accentColor,
                        inactiveColor: Colors.grey,
                        onChanged: (value) {
                          setState(() => tempPhaseDuration =
value.toInt());
                        },
                      ),
                    ],
                  ),
                  Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      const Text(
                        'Зберігати значення?',
                        style: TextStyle(color: Colors.white),
                      ),
                      IconButton(
                        icon: Icon(
                          retain
                              ? Icons.check_box
                              : Icons.check_box_outline_blank,
                          color: retain ? accentColor :
Colors.grey,
                        ),
                        onPressed: () {
                          setState(() {
                            retain = !retain;
                          });
                        },
                      ),
                    ],
                  ),
                ],
              ),
              actions: [
                TextButton(
                  onPressed: () => Navigator.pop(context),
                  child: const Text(
                    "Скасувати",
                    style: TextStyle(color: Colors.red),
                  ),
                ),
                TextButton(
                  onPressed: () async {
                    String phaseColors =
                        "$tempRed1 $tempGreen1 $tempBlue1
$tempRed2 $tempGreen2 $tempBlue2 $tempPhaseDuration";
```

```
                    await firestore.FirebaseFirestore.instance
                        .collection('MQTT')
                        .doc('lightData')
                        .update({
                      'isPhaseActive': 1,
                      'phaseColors': phaseColors,
                    });
                    sendLightPhaseIsActiveToMQTT(isPhaseActive);

                    await sendLightPhaseToFirebaseAndMQTT(
                      tempRed1,
                      tempGreen1,
                      tempBlue1,
                      tempRed2,
                      tempGreen2,
                      tempBlue2,
                      tempPhaseDuration,
                    );

                    setState(() {
                      red1 = tempRed1;
                      green1 = tempGreen1;
                      blue1 = tempBlue1;
                      red2 = tempRed2;
                      green2 = tempGreen2;
                      blue2 = tempBlue2;
                      phaseDuration = tempPhaseDuration;
                      isPhaseActive = true;
                    });

                    Navigator.pop(context);
                  },
                  child: const Text(
                    "Зберегти",
                    style: TextStyle(color: Colors.green),
                  ),
                ),
              ],
            );
          },
        );
      },
    );
  }

  Widget _buildColorSlider(
      String label, int value, ValueChanged<int> onChanged) {
    return Column(
      children: [
        Text(label, style: const TextStyle(color: Colors.white)),
        Slider(
          value: value.toDouble(),
```

```
            min: 0,
            max: 255,
            divisions: 255,
            activeColor: accentColor,
            inactiveColor: Colors.grey,
            onChanged: (newValue) => onChanged(newValue.toInt()),
          ),
        ],
      );
    }

  Future<void> sendLightPhaseIsActiveToMQTT(bool isPhaseActive)
async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic =
'BerezaOleksandr/berezaLight/lightData/isPhaseActive';
    final payload = '$isPhaseActive';
    await _loadSettings();

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      client.logging(on: true);
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;

      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();

        if (client.connectionStatus?.state ==
MqttConnectionState.connected) {
          print('Підключено до $brokerIp!');
          return true;
        } else {
          print('Стан підключення:
${client.connectionStatus?.state}');
        }
      } catch (e) {
        print('Помилка підключення до $brokerIp: $e');
      }
      return false;
    }

    final client = MqttServerClient(localBrokerIp, clientId);

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }
```

```
    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(
      topic,
      MqttQos.atLeastOnce,
      builder.payload!,
    );

    print('Дані успішно відправлено: $payload');

    client.disconnect();
  }

  Future<void> _togglePhaseMode() async {
    int newPhaseState = isPhaseActive ? 0 : 1;

    await firestore.FirebaseFirestore.instance
        .collection('MQTT')
        .doc('lightData')
        .update({
      'isPhaseActive': newPhaseState,
    });

    setState(() {
      isPhaseActive = newPhaseState == 1;
    });
  }

  Future<File> _getPresetsFile() async {
    final dir = await getApplicationDocumentsDirectory();
    return File('${dir.path}/presets.json');
  }

  Future<List<Map<String, dynamic>>> loadPresets() async {
    try {
      final file = await _getPresetsFile();
      if (!await file.exists()) return [];
      final content = await file.readAsString();
      return List<Map<String,
dynamic>>.from(json.decode(content));
    } catch (e) {
      return [];
    }
  }

  Future<void> savePresets(List<Map<String, dynamic>> presets)
async {
    final file = await _getPresetsFile();
    await file.writeAsString(json.encode(presets));
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color.fromARGB(255, 10, 10, 10),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Expanded(
            child: Center(
              child: isPhaseActive
                  ? Column(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: [
                        GestureDetector(
                          onTap: _showLightPhaseDialog,
                          child: Container(
                            width: 180,
                            height: 180,
                            decoration: BoxDecoration(
                              gradient: LinearGradient(
                                colors: [
                                  Color.fromARGB(255, red1,
green1, blue1),
                                  Color.fromARGB(255, red2,
green2, blue2),
                                ],
                                begin: Alignment.topLeft,
                                end: Alignment.bottomRight,
                              ),
                              shape: BoxShape.circle,
                            ),
                          ),
                        ),
                        const SizedBox(height: 32),
                        const Text(
                          "Свайпніть вліво або вправо, щоб змінити
кольори",
                          style: TextStyle(color: Colors.white60,
fontSize: 14),
                          textAlign: TextAlign.center,
                        ),
                        const SizedBox(height: 56),
                      ],
                    )
                  : Column(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: [
                        GestureDetector(
                          onTap: _showColorPickerDialog,
                          child: LightStatusWidget(
                            red: red,
```

```
                            green: green,
                            blue: blue,
                          ),
                        ),
                        const SizedBox(height: 16),
                        Padding(
                          padding: const EdgeInsets.all(16.0),
                          child: ElevatedButton(
                            onPressed: () {
                              Navigator.push(
                                context,
                                MaterialPageRoute(
                                    builder: (context) =>
PresetPage(
                                          onPresetActivated: (r,
g, b) async {
                                            bool retain1 = retain;
                                            retain = true;
                                            await
sendLightToMQTT(r, g, b);
                                            retain = retain1;
                                            refresh();
                                          },
                                        )),
                              );
                            },
                            style: ElevatedButton.styleFrom(
                              backgroundColor:
accentColor.withAlpha(230),
                              padding: const EdgeInsets.symmetric(
                                  vertical: 16, horizontal: 8),
                              textStyle: const TextStyle(
                                  fontSize: 16, color:
Colors.white),
                            ),
                            child: const Text(
                              "Переглянути пресети",
                              textAlign: TextAlign.center,
                              style:
                                  TextStyle(fontSize: 16, color:
Colors.white),
                            ),
                          ),
                        ),
                      ],
                    ),
                  ),
                ),
                SwitchListTile(
                  title: const Text(
                    'Автоматичне освітлення',
                    style: TextStyle(color: Colors.white),
```

```
            ),
            activeColor: accentColor,
            value: isAutoLightMode,
            onChanged: (val) async {
              setState(() => isAutoLightMode = val);
              await sendAutoLightModeToMQTT(val);
            },
          ),
          const SizedBox(height: 15),

          Padding(
            padding: const EdgeInsets.all(16.0),
            child: ElevatedButton(
              onPressed: () async {
                await _togglePhaseMode();
              },
              style: ElevatedButton.styleFrom(
                backgroundColor: accentColor.withAlpha(230),
                padding:
                    const EdgeInsets.symmetric(vertical: 16,
horizontal: 8),
                textStyle: const TextStyle(fontSize: 16, color:
Colors.white),
              ),
              child: Text(
                isPhaseActive
                    ? "Перемкнути на режим одного кольору"
                    : "Перемкнути на режим переливання",
                textAlign: TextAlign.center,
                style: const TextStyle(fontSize: 16, color:
Colors.white),
              ),
            ),
          ),
        ],
      ),
    );
  }
}

class PresetPage extends StatefulWidget {
  final Function(int, int, int) onPresetActivated;

  const PresetPage({super.key, required this.onPresetActivated});

  @override
  State<PresetPage> createState() => _PresetPageState();
}

class _PresetPageState extends State<PresetPage> {
  List<Map<String, dynamic>> _presets = [];
  Color accentColor = purpleAccentColor;
```

```
  @override
  void initState() {
    super.initState();
    _loadSettings();
    _loadPresets();
  }

  Future<File> _getPresetsFile() async {
    final dir = await getApplicationDocumentsDirectory();
    return File('${dir.path}/presets.json');
  }

  Future<void> _loadSettings() async {
    final settings = await SettingsManager.loadSettings();
    setState(() {
      accentColor = settings['accentColor'];
    });
  }

  Future<void> _loadPresets() async {
    final file = await _getPresetsFile();
    if (!await file.exists()) return;

    final content = await file.readAsString();
    setState(() {
      _presets = List<Map<String,
dynamic>>.from(json.decode(content));
    });
  }

  Future<void> _savePresets() async {
    final file = await _getPresetsFile();
    await file.writeAsString(json.encode(_presets));
  }

  void _addOrEditPreset({Map<String, dynamic>? preset, int?
index}) {
    int r = preset?['r'] ?? 128;
    int g = preset?['g'] ?? 0;
    int b = preset?['b'] ?? 255;
    String name = preset?['name'] ?? '';

    showDialog(
      context: context,
      builder: (ctx) {
        return StatefulBuilder(
          builder: (ctx, setState) {
            return AlertDialog(
              title:
                  Text(index == null ? 'Новий пресет' :
'Редагування пресету'),
```

```
                content: SingleChildScrollView(
                  child: Column(
                    mainAxisSize: MainAxisSize.min,
                    children: [
                      Container(
                        width: 80,
                        height: 80,
                        decoration: BoxDecoration(
                          color: Color.fromARGB(255, r, g, b),
                          shape: BoxShape.circle,
                          border: Border.all(color: Colors.grey,
width: 2),
                        ),
                      ),
                      TextField(
                        controller: TextEditingController(text:
name),
                        decoration: const InputDecoration(labelText:
'Назва'),
                        onChanged: (val) => name = val,
                      ),
                      const SizedBox(height: 16),
                      _buildSlider("Red", r, (val) => setState(() =>
r = val)),
                      _buildSlider("Green", g, (val) => setState(()
=> g = val)),
                      _buildSlider("Blue", b, (val) => setState(()
=> b = val)),
                      if (index != null)
                        TextButton(
                          onPressed: () {
                            setState(() {
                              _presets.removeAt(index);
                            });
                            _savePresets();
                            Navigator.pop(context);
                            _loadPresets();
                          },
                          child: const Text("Видалити",
                              style: TextStyle(color: Colors.red)),
                        )
                    ],
                  ),
                ),
                actions: [
                  TextButton(
                    child: const Text("Скасувати"),
                    onPressed: () => Navigator.pop(context),
                  ),
                  TextButton(
                    child: const Text("Зберегти",
                        style: TextStyle(color: Colors.green)),
```

```
                    onPressed: () {
                      final newPreset = {"name": name, "r": r, "g":
g, "b": b};
                      setState(() {
                        if (index == null) {
                          _presets.add(newPreset);
                        } else {
                          _presets[index] = newPreset;
                        }
                      });
                      _savePresets();
                      Navigator.pop(context);
                      _loadPresets();
                    },
                  ),
                ],
              );
            },
          );
        },
      );
    }

    Widget _buildSlider(String label, int value, Function(int)
onChanged) {
      return Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text('$label: $value'),
          Slider(
            value: value.toDouble(),
            min: 0,
            max: 255,
            divisions: 255,
            activeColor: accentColor,
            inactiveColor: Colors.grey,
            onChanged: (val) => onChanged(val.toInt()),
          ),
        ],
      );
    }

    @override
    Widget build(BuildContext context) {
      return Scaffold(
        backgroundColor: const Color.fromARGB(255, 10, 10, 10),
        appBar: AppBar(
          title: const Text("Пресети світла"),
          backgroundColor: Color.fromARGB(255, 20, 18, 24),
          actions: [
            IconButton(
              icon: const Icon(Icons.add, color: Colors.green),
```

```
            onPressed: () => _addOrEditPreset(),
          )
        ],
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: _presets.isEmpty
            ? const Center(
                child: Text("Немає пресетів",
                    style: TextStyle(color: Colors.white70)))
            : Wrap(
                spacing: 16,
                runSpacing: 16,
                children: _presets.asMap().entries.map((entry) {
                  final idx = entry.key;
                  final preset = entry.value;
                  final color = Color.fromARGB(
                      255, preset['r'], preset['g'], preset['b']);

                  return GestureDetector(
                    onTap: () {
                      widget.onPresetActivated(
                          preset['r'], preset['g'], preset['b']);
                      Navigator.pop(context);
                    },
                    onLongPress: () {
                      _addOrEditPreset(preset: preset, index:
idx);
                    },
                    child: Column(
                      mainAxisSize: MainAxisSize.min,
                      children: [
                        CircleAvatar(
                          radius: 30,
                          backgroundColor: color,
                        ),
                        const SizedBox(height: 6),
                        Text(
                          preset['name'] ?? '',
                          style: const TextStyle(color:
Colors.white),
                        )
                      ],
                    ),
                  );
                }).toList(),
              ),
      ),
    );
  }
}
```

```
class SensorDataWidget extends StatelessWidget {
  final IconData icon;
  final Color iconColor;
  final String label;
  final String value;

  const SensorDataWidget({
    super.key,
    required this.icon,
    required this.iconColor,
    required this.label,
    required this.value,
  });

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          icon,
          color: iconColor,
          size: 22,
        ),
        const SizedBox(width: 10),
        Text(
          "$label: $value",
          style: const TextStyle(color: Colors.white),
        ),
      ],
    );
  }
}

class LightStatusWidget extends StatelessWidget {
  final int red;
  final int green;
  final int blue;

  const LightStatusWidget({
    super.key,
    required this.red,
    required this.green,
    required this.blue,
  });

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Container(
          width: 180,
```

```
            height: 180,
            decoration: BoxDecoration(
              color: Color.fromRGBO(red, green, blue, 1),
              shape: BoxShape.circle,
              border: Border.all(color: Colors.white30, width: 2),
            ),
          ),
          const SizedBox(height: 10),
        ],
      );
    }
}

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Light control',
      theme: ThemeData.dark().copyWith(
        scaffoldBackgroundColor:
            const Color.fromARGB(255, 10, 10, 10),
        primaryColor: purpleAccentColor, // Акцентний колір
        appBarTheme: const AppBarTheme(
          backgroundColor: Color.fromARGB(255, 18, 18, 18), //
Темна панель
          titleTextStyle: TextStyle(color: Colors.white, fontSize:
20),
        ),
      ),
      debugShowCheckedModeBanner: false,
      home: const MainPage(),
    );
  }
}

class SensorInfoScreen extends StatefulWidget {
  const SensorInfoScreen({super.key});

  @override
  _SensorInfoScreenState createState() =>
_SensorInfoScreenState();
}
```

```
double feelsLikeTemp = 0.0;
double indoorTemperature = 0.0;
int indoorHumidity = 0;
double outdoorTemperature = 0.0;
int outdoorHumidity = 0;
double windSpeed = 0.0;
int windDegree = 0;
String weatherDescription = "Завантаження...";
String weatherIcon = "01d";
int mq9Data = 0;
int gl5528Data = 0;

class _SensorInfoScreenState extends State<SensorInfoScreen> {
  String localBrokerIp = '192.168.0.106';
  String vpnBrokerIp = '192.168.0.144';
  Color accentColor = purpleAccentColor;
  Color windSpeedColor = const Color.fromARGB(255, 128, 140, 146);
  IconData weatherDescriptionIcon = WeatherIcons.cloud_refresh;
  late Timer _timer;

  @override
  void initState() {
    super.initState();
    _fetchSensorData();
    _fetchExternalWeather();
    _loadSettings();

    // Оновлення даних кожні 5 хвилин
    _timer = Timer.periodic(const Duration(minutes: 2), (timer) {
      if (mounted) {
        _fetchSensorData();
        _fetchExternalWeather();
      }
    });
    SystemChrome.setSystemUIOverlayStyle(const
SystemUiOverlayStyle(
      systemNavigationBarColor:
          Color.fromARGB(255, 20, 18, 24),
      systemNavigationBarIconBrightness:
          Brightness.light,
    ));
  }

  Future<void> _loadSettings() async {
    final settings = await SettingsManager.loadSettings();
    setState(() {
      localBrokerIp = settings['localIp'];
      vpnBrokerIp = settings['vpnIp'];
      accentColor = settings['accentColor'];
    });
  }
```

```
  Future<void> _fetchSensorData() async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topicDHT = 'Devices/ESP01_DHT';
    const topicLIGHT = 'Devices/D1_GL5528';
    const topicMQ9 = 'Devices/mq9';

    await _loadSettings();

    final client = MqttServerClient(localBrokerIp, clientId);
    client.logging(on: true);
    client.keepAlivePeriod = 6;
    client.onDisconnected = () => print('🔌 Відключено від MQTT');

    Future<bool> tryConnect(String ip) async {
      client.server = ip;
      try {
        await client.connect();
        return client.connectionStatus?.state ==
MqttConnectionState.connected;
      } catch (e) {
        print('❌ Помилка підключення до $ip: $e');
        return false;
      }
    }

    if (!await tryConnect(localBrokerIp)) {
      if (!await tryConnect(vpnBrokerIp)) {
        print('❌ Не вдалося підключитись до жодного брокера');
        return;
      }
    }

    final dhtCompleter = Completer<void>();
    final mq9Completer = Completer<void>();
    final lightCompleter = Completer<void>();

    client.subscribe(topicDHT, MqttQos.atLeastOnce);
    client.subscribe(topicMQ9, MqttQos.atLeastOnce);
    client.subscribe(topicLIGHT, MqttQos.atLeastOnce);

    client.updates!.listen((List<MqttReceivedMessage<MqttMessage>>
messages) {
      final recMsg = messages[0].payload as MqttPublishMessage;
      final topic = messages[0].topic;
      final payload =

MqttPublishPayload.bytesToStringAsString(recMsg.payload.message);

      print('📩 MQTT повідомлення: [$topic] $payload');
```

```
      if (topic == topicDHT) {
        final parts = payload.split(';');
        if (parts.length == 2) {
          final temp = double.tryParse(parts[0].trim());
          final hum = int.tryParse(parts[1].trim());
          if (temp != null && hum != null) {
            setState(() {
              indoorTemperature = temp;
              indoorHumidity = hum;
            });
            dhtCompleter.complete();
          }
        }
      } else if (topic == topicMQ9) {
        final mq9 = int.tryParse(payload.trim());
        if (mq9 != null) {
          setState(() => mq9Data = mq9);
          mq9Completer.complete();
        }
      } else if (topic == topicLIGHT) {
        final light = int.tryParse(payload.trim());
        if (light != null) {
          setState(() => gl5528Data = light);
          lightCompleter.complete();
        }
      }
    });

    await Future.wait([
      dhtCompleter.future.timeout(
        const Duration(seconds: 5),
        onTimeout: () => print('⚠ Таймаут зчитування з
$topicDHT'),
      ),
      mq9Completer.future.timeout(
        const Duration(seconds: 5),
        onTimeout: () => print('⚠ Таймаут зчитування з
$topicMQ9'),
      ),
      lightCompleter.future.timeout(
        const Duration(seconds: 5),
        onTimeout: () => print('⚠ Таймаут зчитування з
$topicLIGHT'),
      ),
    ]);

    client.disconnect();
  }

  Future<void> _fetchExternalWeather() async {
    const apiKey = '958e24c188736bcd2ab346f85ec87630';
```

```
    const city = 'ternopil';
    const url =

'https://api.openweathermap.org/data/2.5/weather?q=$city&units=met
ric&appid=$apiKey&lang=UA';

    try {
      final response = await http.get(Uri.parse(url));
      if (response.statusCode == 200) {
        final data = json.decode(response.body);
        setState(() {
          outdoorTemperature = data['main']['temp'];
          outdoorHumidity = data['main']['humidity'];
          feelsLikeTemp = data['main']['feels_like'];
          windSpeed = data['wind']['speed'];
          windDegree = data['wind']['deg'];
          weatherDescription =

_capitalizeFirstLetter(data['weather'][0]['description']);
          weatherIcon = data['weather'][0]['icon'];

          firestore.FirebaseFirestore.instance
              .collection('MQTT')
              .doc('SensorData')
              .update({
            'outdoorTemperature': outdoorTemperature,
            'outdoorHumidity': outdoorHumidity,
            'feelsLikeTemp': feelsLikeTemp,
            'windSpeed': windSpeed,
            'windDegree': windDegree,
            'weatherDescription': weatherDescription,
            'weatherIcon': weatherIcon,
            'time': firestore.Timestamp.now(),
          });

          weatherDescriptionIcon =
_getWeatherIcon(weatherDescription);
        });
      } else {
        print('Помилка завантаження погоди:
${response.statusCode}');
      }
    } catch (e) {
      print('Помилка: $e');
    }

    if (windSpeed >= 12 && windSpeed < 17) {
      windSpeedColor = const Color.fromARGB(255, 224, 222, 76);
    } else if (windSpeed >= 17 && windSpeed < 24) {
      windSpeedColor = const Color.fromARGB(255, 224, 155, 76);
    } else if (windSpeed >= 24) {
      windSpeedColor = const Color.fromARGB(255, 224, 76, 76);
```

```
    } else {
      windSpeedColor = const Color.fromARGB(255, 255, 255, 255);
    }
  }

  IconData _getWeatherIcon(String description) {
    switch (description.toLowerCase()) {
      case "ясно":
      case "чисте небо":
        return WeatherIcons.day_sunny;

      case "невелика хмарність":
      case "мінлива хмарність":
      case "рвані хмари":
      case "уривчасті хмари":
        return WeatherIcons.cloud;

      case "хмарно":
      case "сильна хмарність":
        return WeatherIcons.cloudy;

      case "дощ":
      case "злива":
        return WeatherIcons.rain;

      case "гроза":
        return WeatherIcons.thunderstorm;

      case "сніг":
        return WeatherIcons.snow;

      case "туман":
        return WeatherIcons.fog;

      default:
        return WeatherIcons.cloudy;
    }
  }

  String _capitalizeFirstLetter(String text) {
    if (text.isEmpty) return text;
    return text[0].toUpperCase() + text.substring(1);
  }

  Color getTemperatureColor(double temp) {
    if (temp <= -5) return Colors.blueAccent;
    if (temp >= 20) return Colors.orangeAccent;
    double ratio = temp / 25;
    return Color.lerp(Colors.blueAccent, Colors.orangeAccent,
ratio)!;
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color.fromARGB(255, 10, 10, 10),
      body: Padding(
        padding: const EdgeInsets.symmetric(vertical: 0.0,
horizontal: 16.0),
        child: SingleChildScrollView(
          physics: const BouncingScrollPhysics(),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              const SizedBox(height: 40),
              Container(
                decoration: BoxDecoration(
                  color: Colors.white10,
                  borderRadius: BorderRadius.circular(12),
                ),
                padding: const EdgeInsets.all(16),
                child: Row(
                  children: [
                    Icon(weatherDescriptionIcon, size: 64, color:
accentColor),
                    const SizedBox(width: 54),
                    Text(
                      textAlign: TextAlign.center,
                      weatherDescription,
                      style: const TextStyle(fontSize: 18, color:
Colors.white),
                    ),
                  ],
                ),
              ),
              const SizedBox(height: 20),

              Row(
                children: [
                  _sensorCard(
                    icon: WeatherIcons.thermometer,
                    color: getTemperatureColor(indoorTemperature),
                    label: "Температура в приміщенні",
                    value: "$indoorTemperature°C",
                  ),
                  const SizedBox(width: 12),
                  _sensorCard(
                    icon: WeatherIcons.humidity,
                    color: Colors.blueAccent,
                    label: "Вологість в приміщенні",
                    value: "$indoorHumidity%",
                  ),
                ],
              ),
```

```
                const SizedBox(height: 12),

                Row(
                  children: [
                    _sensorCard(
                      icon: WeatherIcons.dust,
                      color: Colors.white,
                      label: "Присутність шкідливих газів",
                      value: "$mq9Data",
                    ),
                    const SizedBox(width: 12),
                    _sensorCard(
                      icon: WeatherIcons.moon_new,
                      color: Colors.white,
                      label: "Освітленість навулиці",
                      value: "$gl5528Data",
                    ),
                  ],
                ),

                const SizedBox(height: 12),

                Row(
                  children: [
                    _sensorCard(
                      icon: WeatherIcons.thermometer,
                      color:
getTemperatureColor(outdoorTemperature),
                      label: "Температура навулиці",
                      value: "$outdoorTemperature°C",
                    ),
                    const SizedBox(width: 12),
                    _sensorCard(
                      icon: WeatherIcons.humidity,
                      color: Colors.blueAccent,
                      label: "Вологість навулиці",
                      value: "$outdoorHumidity%",
                    ),
                  ],
                ),
                const SizedBox(height: 12),

                Row(children: [
                  _sensorCard(
                    icon: WeatherIcons.thermometer_exterior,
                    color: getTemperatureColor(feelsLikeTemp),
                    label: "Відчувається як",
                    value: "$feelsLikeTemp°C",
                  ),
                ]),
                const SizedBox(height: 12),
                Row(
```

```
                children: [
                  _sensorCard(
                    icon: WeatherIcons.strong_wind,
                    color: windSpeedColor,
                    label: "Швидкість вітру",
                    value: "$windSpeed м/с",
                  ),
                  const SizedBox(width: 12),
                  _sensorCard(
                    icon: WeatherIcons.wind_direction,
                    color: Colors.white,
                    label: "Напрямок вітру",
                    value: "$windDegree°",
                    rotate: windDegree.toDouble(),
                  ),
                ],
              ),
              const SizedBox(height: 16),
            ],
          ),
        ),
      ),
    );
  }

  Widget _sensorCard(
      {required IconData icon,
      required Color color,
      required String label,
      required String value,
      double rotate = 0}) {
    return Expanded(
      child: Container(
        decoration: BoxDecoration(
          color: Colors.white10,
          borderRadius: BorderRadius.circular(12),
        ),
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            Transform.rotate(
              angle: rotate * (3.1415926535 / 180),
              child: Icon(icon, size: 36, color: color),
            ),
            const SizedBox(height: 8),
            Text(
              label,
              textAlign: TextAlign.center,
              style: const TextStyle(fontSize: 14, color:
Colors.grey),
            ),
            const SizedBox(height: 4),
```

```
            Text(
              value,
              style: const TextStyle(fontSize: 18, color:
Colors.white),
            ),
          ],
        ),
      ),
    );
  }

  @override
  void dispose() {
    _timer.cancel();
    super.dispose();
  }
}

class MainPage extends StatefulWidget {
  const MainPage({Key? key}) : super(key: key);

  @override
  State<MainPage> createState() => _MainPageState();
}

class _MainPageState extends State<MainPage> {
  int _selectedIndex = 0;
  Color accentColor = purpleAccentColor;

  Future<void> _loadSettings() async {
    final settings = await SettingsManager.loadSettings();
    setState(() {
      accentColor = settings['accentColor'];
    });
  }

  final List<Widget> _pages = [
    const LightScreen(),
    const SensorInfoScreen(),
    const SpeechScreen(),
    const AutomationPage(),
    const FirebaseScreen(),
    const SettingsScreen(),
  ];

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
      _loadSettings();
    });
  }
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: _pages[_selectedIndex],
    bottomNavigationBar: BottomNavigationBar(
      backgroundColor: const Color.fromARGB(255, 18, 18, 18),
      selectedItemColor: accentColor,
      unselectedItemColor: Colors.grey,
      items: [
        BottomNavigationBarItem(
          icon: _glowingIcon(Icons.lightbulb, 0),
          label: 'Світло',
        ),
        BottomNavigationBarItem(
          icon: _glowingIcon(Icons.sensors, 1),
          label: 'Датчики',
        ),
        BottomNavigationBarItem(
          icon: _glowingIcon(Icons.keyboard, 2),
          label: 'Керування',
        ),
        BottomNavigationBarItem(
          icon: _glowingIcon(Icons.adb_rounded, 3),
          label: 'Автоматизація',
        ),
        BottomNavigationBarItem(
          icon: _glowingIcon(Icons.storage, 4),
          label: 'База даних',
        ),
        BottomNavigationBarItem(
          icon: _glowingIcon(Icons.settings, 5),
          label: 'Налаштування',
        ),
      ],
      currentIndex: _selectedIndex,
      onTap: _onItemTapped,
    ),
  );
}

Widget _glowingIcon(IconData icon, int index) {
  bool isSelected = _selectedIndex == index;

  return Container(
    decoration: BoxDecoration(
      shape: BoxShape.circle,
      boxShadow: isSelected
          ? [
              BoxShadow(
                color: accentColor.withOpacity(0.4),
                blurRadius: 12,
                spreadRadius: 2,
```

```
                ),
              ]
            : [],
      ),
      child:
          Icon(icon, size: 24, color: isSelected ? accentColor :
Colors.grey),
    );
  }
}

class SpeechScreen extends StatefulWidget {
  const SpeechScreen({Key? key}) : super(key: key);
  // This class is the configuration for the state. It holds the
values (in this
  // case the title) provided by the parent (in this case the App
widget) and
  // used by the build method of the State. Fields in a Widget
subclass are
  // always marked "final".

  final String title = 'Speech Recognition';

  @override
  State<SpeechScreen> createState() => _SpeechScreenState();
}

class _SpeechScreenState extends State<SpeechScreen> {
  String localBrokerIp = '192.168.0.106';
  String vpnBrokerIp = '192.168.0.144';
  final TextEditingController _textController =
TextEditingController();
  bool _isTextFieldActive = false;
  bool isRecording = false;
  final String apiKey =
      'sk-proj-m6mdrabBOhm8f9YEeBMPp8ebUGPmr9KxTHRr9yPy-
Ykmh7KrRcjj9I87KDufsLwt8zVF33t-7VT3BlbkFJ1we0WOyx-
fTfpizn10tFoU6aX8dwbF7FSjlG_Ng4mPGk8zTdmUGQcswEz-
PJ8Qyq7qG_ICNDcA';
  String ttsApiKey = 'AIzaSyB1otn2UsNrCBeive4HUhHTRE9MT8OF780';
  String responseTextFull = '';
  String responseCommandFull = '';
  late SpeechToText speech;
  late TexttospeechApi textToSpeech;

  AudioPlayer audioPlayer = AudioPlayer();
  bool _shouldSpeak = true;
  Color micColor = Colors.white;
  Color accentColor = purpleAccentColor;
  Color btnBgCol = const Color.fromARGB(255, 105, 73, 136);
  int icCol = 18;
  int red = 255;
```

```
  int green = 255;
  int blue = 255;
  int red1 = 255;
  int green1 = 255;
  int blue1 = 255;
  int red2 = 50;
  int green2 = 50;
  int blue2 = 50;
  int phaseDuration = 1;
  bool isPhaseActive = false;
  bool chatGPTAnswerPulse = true;
  bool isAutoMode = true;
  int outdoorLightValue = 0;
  double indoorTemperature = 0;
  int indoorHumidity = 0;
  int indoorMQ9 = 0;

  final _record =
      RecordPlatform.instance;
  final config = RecognitionConfig(
      encoding: AudioEncoding.LINEAR16,
      model: RecognitionModel.basic,
      enableAutomaticPunctuation: true,
      sampleRateHertz: 16000,
      languageCode: 'uk-UA');

  @override
  void initState() {
    super.initState();
    _loadSettings();
    _requestMicrophonePermission();
    initializeTextToSpeech();
    _loadLightColor();
    initializeRecorder();
    initializeSpeech();
    _fetchExternalWeather();
    _fetchSensorData();
    SystemChrome.setSystemUIOverlayStyle(const
SystemUiOverlayStyle(
      systemNavigationBarColor:
          Color.fromARGB(255, 20, 18, 24),
      systemNavigationBarIconBrightness:
          Brightness.light,
    ));
  }

  Future<void> _fetchSensorData() async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topicDHT = 'Devices/ESP01_DHT';
    const topicLIGHT = 'Devices/D1_GL5528';
    const topicMQ9 = 'Devices/mq9';
```

```
    await _loadSettings();

    final client = MqttServerClient(localBrokerIp, clientId);
    client.logging(on: true);
    client.keepAlivePeriod = 6;
    client.onDisconnected = () => print('🔌 Відключено від MQTT');

    Future<bool> tryConnect(String ip) async {
      client.server = ip;
      try {
        await client.connect();
        return client.connectionStatus?.state ==
MqttConnectionState.connected;
      } catch (e) {
        print('❌ Помилка підключення до $ip: $e');
        return false;
      }
    }

    if (!await tryConnect(localBrokerIp)) {
      if (!await tryConnect(vpnBrokerIp)) {
        print('❌ Не вдалося підключитись до жодного брокера');
        return;
      }
    }

    final dhtCompleter = Completer<void>();
    final mq9Completer = Completer<void>();
    final lightCompleter = Completer<void>();


    client.subscribe(topicDHT, MqttQos.atLeastOnce);
    client.subscribe(topicMQ9, MqttQos.atLeastOnce);
    client.subscribe(topicLIGHT, MqttQos.atLeastOnce);

    client.updates!.listen((List<MqttReceivedMessage<MqttMessage>>
messages) {
      final recMsg = messages[0].payload as MqttPublishMessage;
      final topic = messages[0].topic;
      final payload =

MqttPublishPayload.bytesToStringAsString(recMsg.payload.message);

      print('📩 MQTT повідомлення: [$topic] $payload');

      if (topic == topicDHT) {
        final parts = payload.split(';');
        if (parts.length == 2) {
          final temp = double.tryParse(parts[0].trim());
          final hum = int.tryParse(parts[1].trim());
          if (temp != null && hum != null) {
```

```
            setState(() {
              indoorTemperature = temp;
              indoorHumidity = hum;
            });
            dhtCompleter.complete();
          }
        }
      } else if (topic == topicMQ9) {
        final mq9 = int.tryParse(payload.trim());
        if (mq9 != null) {
          setState(() -> indoorMQ9 - mq9);
          mq9Completer.complete();
        }
      } else if (topic == topicLIGHT) {
        final light = int.tryParse(payload.trim());
        if (light != null) {
          setState(() => outdoorLightValue = light);
          lightCompleter.complete();
        }
      }
    });

    await Future.wait([
      dhtCompleter.future.timeout(
        const Duration(seconds: 5),
        onTimeout: () => print('⚠ Таймаут зчитування з
$topicDHT'),
      ),
      mq9Completer.future.timeout(
        const Duration(seconds: 5),
        onTimeout: () => print('⚠ Таймаут зчитування з
$topicMQ9'),
      ),
      lightCompleter.future.timeout(
        const Duration(seconds: 5),
        onTimeout: () => print('⚠ Таймаут зчитування з
$topicLIGHT'),
      ),
    ]);

    client.disconnect();
  }

  Future<void> initializeSpeech() async {
    final serviceAccount = ServiceAccount.fromString(
      await DefaultAssetBundle.of(context)
          .loadString('assets/service_account.json'),
    );
    speech = SpeechToText.viaServiceAccount(serviceAccount);
  }
```

```
  Future<void> initializeTextToSpeech() async {
    final client = http.Client();
    textToSpeech =
        TexttospeechApi(clientViaApiKey(ttsApiKey, baseClient:
client));
  }

  Future<void> initializeRecorder() async {
    await _record.create('my_recorder');
    if (await _record.hasPermission('my_recorder')) {
      print('Рекордер готовий до використання');
    } else {
      print('Немає дозволу на використання мікрофона');
    }
  }

  Future<void> startRecording() async {
    if (await Permission.microphone.request().isGranted) {
      final directory = await getApplicationDocumentsDirectory();
      final audioFilePath =
          '${directory.path}/audio.wav';

      await _record.start(
        'my_recorder',
        const RecordConfig(
          encoder: AudioEncoder.wav,
          sampleRate: 16000,
          numChannels: 1,
          noiseSuppress: true,
        ),
        path: audioFilePath,
      );

      setState(() {
        isRecording = true;
        reqSendState = '';
        recognizedText = '';
        recognizedTextFull = '';
        userInputHintText = 'Очікування закінчення запису';
        micColor = Colors.white;
        btnBgCol = const Color.fromARGB(255, 199, 54, 54);
        icCol = 200;
      });
    } else {
      recognizedText = 'Мікрофон недоступний.';
      print('Мікрофон недоступний');
    }
  }

  Future<void> _loadLightColor() async {
    var lightDoc = await firestore.FirebaseFirestore.instance
        .collection('MQTT')
```

```
          .doc('lightData')
          .get();

      if (lightDoc.exists) {
        var lightData = lightDoc.data()!;
        setState(() {
          isPhaseActive = lightData['isPhaseActive'] == 1;
          String phaseColors = lightData['phaseColors'] ?? '';
          List<String> colorParts = phaseColors.split(' ');

          if (colorParts.length == 7) {
            red1 = int.tryParse(colorParts[0]) ?? 0;
            green1 = int.tryParse(colorParts[1]) ?? 0;
            blue1 = int.tryParse(colorParts[2]) ?? 0;

            red2 = int.tryParse(colorParts[3]) ?? 0;
            green2 = int.tryParse(colorParts[4]) ?? 0;
            blue2 = int.tryParse(colorParts[5]) ?? 0;

            phaseDuration = int.tryParse(colorParts[6]) ?? 1;
            phaseDuration = phaseDuration.clamp(1, 120); //
Обмеження 1-120 секунд
          } else {
            print("Неправильний формат phaseColors: $phaseColors");
          }

          String colorData = lightData['colors'] ?? '255 255 255';
          List<String> singleColorParts = colorData.split(' ');

          red = int.tryParse(singleColorParts[0]) ?? 255;
          green = int.tryParse(singleColorParts[1]) ?? 255;
          blue = int.tryParse(singleColorParts[2]) ?? 255;
        });
      }
    }

    Future<void> _fetchExternalWeather() async {
      const apiKey = '958e24c188736bcd2ab346f85ec87630';
      const city = 'ternopil';
      const url =

'https://api.openweathermap.org/data/2.5/weather?q=$city&units=met
ric&appid=$apiKey&lang=UA';

      try {
        final response = await http.get(Uri.parse(url));
        if (response.statusCode == 200) {
          final data = json.decode(response.body);
          setState(() {
            outdoorTemperature = data['main']['temp'];
            outdoorHumidity = data['main']['humidity'];
            feelsLikeTemp = data['main']['feels_like'];
```

```
          windSpeed = data['wind']['speed'];
          windDegree = data['wind']['deg'];
          weatherDescription = data['weather'][0]['description'];
          weatherIcon = data['weather'][0]['icon'];
        });
      } else {
        print('Помилка завантаження погоди:
${response.statusCode}');
      }
    } catch (e) {
      print('Помилка: $e');
    }
  }

  Future<String> getChatGPTResponse(String userInput) async {
    await _loadLightColor();

    final url =
Uri.parse('https://api.openai.com/v1/chat/completions');
    final customInstructions = await
SettingsManager.loadCustomInstructions();

    final response = await http.post(
      url,
      headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer $apiKey',
      },
      body: jsonEncode({
        'model': 'gpt-4o-mini',
        'messages': [
          {
            'role': 'system',
            'content':
                '''Відповіді розділяй на дві частини: "CHAT_TEXT"
і "CHAT_COMMAND". Коли користувач запитує змінити колір світла, ти
відповідаєш командою "light" з відповідними значеннями для
кольорів, аргументи команди (red, green, blue). Якщо користувач
запитує включити режим переливання між двома кольорами, ти
відповідаєш командою "lightPhase" з відповідними значеннями для
обох кольорів та часу в секундах від 1 до 120 за який буде
виконано зміну першого кольору на другий та назад до першого
кольору, аргументи команди (red1, green1, blue1, red2, green2,
blue2, phaseDuration). Командна частина завжди повинна міститись в
кінці відповіді. Команда light або lightPhase може міститись лише
один раз за відповідь, та лише одна команда за відповідь. Поточні
налаштування світла: light($red, $green, $blue); lightPhase($red1,
$green1, $blue1, $red2, $green2, $blue2, $phaseDuration). Поточна
погода в місті: температура $outdoorTemperature; вологість
$outdoorHumidity; швидкість вітру $windSpeed; освітленість:
$outdoorLightValue (чим  більше тим темніше); відчувається як
$feelsLikeTemp; загальна погода $weatherDescription. Поточні
```

```
показники в приміщенні: температура $indoorTemperature; вологість
$indoorHumidity; присутність шкідливих газів $indoorMQ9 (чим
більше тим гірше повітря); ${customInstructions.trim().isNotEmpty
? "\nКористувацькі інструкції для вмісту текстових відповідей:
${customInstructions.trim()}" : ""}'''
            },
            {'role': 'user', 'content': userInput}
          ],
          'max_tokens': 1500,
          'temperature': 0.8, // менша температура для чіткіших
відповідей
        }),
      );

      print('Запит до OpenAI: $userInput');
      print('URL запиту: $url');
      print('Заголовки: ${response.headers}');
      print('Тіло відповіді: ${response.body}');

      if (response.statusCode == 200) {
        final Map<String, dynamic> data =
            jsonDecode(utf8.decode(response.bodyBytes));
        String chatText =
data['choices'][0]['message']['content'].trim();
        return chatText;
      } else {
        throw Exception(
            'Помилка при зверненні до ChatGPT:
${response.statusCode}');
      }
    }

    Future<void> saveResponseToFirebase(String chatText) async {
      List<String> splitResponse =
chatText.split(RegExp(r'CHAT_COMMAND:?'));
      String responseTextFull = '';

      if (splitResponse.isNotEmpty) {
        responseTextFull =
            splitResponse[0].replaceFirst(RegExp(r'CHAT_TEXT:?'),
'').trim();
      }
      String responseCommandFull =
          splitResponse.length > 1 ? splitResponse[1].trim() : '';

      final snapshot = await firestore.FirebaseFirestore.instance
          .collection('MQTTResponses')
          .orderBy('timestamp', descending: true)
          .limit(1)
          .get();

      int newId = 1;
```

```
    if (snapshot.docs.isNotEmpty) {
      String lastId = snapshot.docs.first.id;
      newId = int.parse(lastId) + 1;
    }

    await firestore.FirebaseFirestore.instance
        .collection('MQTTResponses')
        .doc('$newId')
        .set({
      'chat_text': responseTextFull,
      'chat_command': responseCommandFull,
      'timestamp': firestore.Timestamp.now(),
    });
  }

  Future<void> _requestMicrophonePermission() async {
    var status = await Permission.microphone.status;
    if (!status.isGranted) {
      await Permission.microphone.request();
    }
  }

  Future<void> speakText(String text, String langCode) async {
    final input = SynthesisInput(text: text);
    final voice = VoiceSelectionParams(languageCode: langCode);
    final audioConfig = AudioConfig(audioEncoding: 'MP3');

    final response = await
textToSpeech.text.synthesize(SynthesizeSpeechRequest(
      input: input,
      voice: voice,
      audioConfig: audioConfig,
    ));

    if (response.audioContent != null) {
      final directory = await getApplicationDocumentsDirectory();
      final path = '${directory.path}/output.mp3';
      final audioBytes = response.audioContentAsBytes;
      await File(path).writeAsBytes(audioBytes);
      print('Audio file generated and saved as $path');
      await audioPlayer.play(DeviceFileSource(path));
      await sendChatGPTAnswearPulseToMQTT(chatGPTAnswerPulse);
    } else {
      print('Помилка генерації аудіо.');
    }
  }

  Future<void> replayAudioResponse() async {
    if (_shouldSpeak) {
      final directory = await getApplicationDocumentsDirectory();
      final path = '${directory.path}/output.mp3';
```

```
      await audioPlayer.play(DeviceFileSource(path));
      await sendChatGPTAnswearPulseToMQTT(chatGPTAnswerPulse);
    }
  }

  Future<void> sendLightToMQTT(int red, int green, int blue) async
{
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic = 'BerezaOleksandr/berezaLight/lightData/colors';
    final payload = '$red $green $blue';
    await _loadSettings();

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      client.logging(on: true);
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;

      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();

        if (client.connectionStatus?.state ==
MqttConnectionState.connected) {
          print('Підключено до $brokerIp!');
          return true;
        } else {
          print('Стан підключення:
${client.connectionStatus?.state}');
        }
      } catch (e) {
        print('Помилка підключення до $brokerIp: $e');
      }
      return false;
    }

    final client = MqttServerClient(localBrokerIp, clientId);

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(
      topic,
      MqttQos.atLeastOnce,
```

```
      builder.payload!,
    );

    print('Дані успішно відправлено: $payload');

    client.disconnect();
  }

  void _onDisconnected() {
    print('Відключено від брокера.');
  }

  void _sendText() async {
    final keyboardText = _textController.text;
    if (keyboardText.isNotEmpty) {
      print('Sending: $keyboardText');
      try {
        final snapshot = await
firestore.FirebaseFirestore.instance
            .collection('MQTTRequests')
            .orderBy('timestamp',
                descending: true)
            .limit(1)
            .get();

        int newId = 1;

        if (snapshot.docs.isNotEmpty) {
          String lastId = snapshot.docs.first.id;
          newId = int.parse(lastId) + 1;
        }

        await firestore.FirebaseFirestore.instance
            .collection('MQTTRequests')
            .doc('$newId')
            .set({
          'text': keyboardText.isNotEmpty ? keyboardText :
"Помилка",
          'timestamp': firestore.Timestamp.now(),
        });

        print("Новий документ з ID $newId додано успішно!");
        reqSendState = "Новий документ з ID $newId додано
успішно!";
      } catch (e) {
        print("Помилка при додаванні нового документа: $e");
        reqSendState = "Помилка при додаванні нового документа.";
      }
      recognizedTextFull = keyboardText;
      userInputHintText = keyboardText;
      await sendToChatGPT();
      await updateCommandInFirestore(responseCommandFull);
```

```
      _textController.clear();
    }
  }

  Future<void> updateCommandInFirestore(String commandString)
async {
    if (commandString.isEmpty ||
        commandString.toLowerCase() == 'none' ||
        commandString.toLowerCase() == 'None' ||
        commandString.toLowerCase().contains('немає команди для
виконання') ||
        commandString.toLowerCase().contains('немає команди') ||
        commandString.toLowerCase().contains('немає')) {
      print('Немає команди для виконання.');
      return;
    }

    final commandName = commandString.split('(')[0].trim();

    try {
      final argsString = commandString.substring(
        commandString.indexOf('(') + 1,
        commandString.indexOf(')'),
      );

      final args = argsString.split(',').map((arg) =>
arg.trim()).toList();

      if (commandName == 'light') {
        if (args.length != 3) {
          throw Exception('Неправильна кількість аргументів для
light');
        }

        final int red = int.parse(args[0]);
        final int green = int.parse(args[1]);
        final int blue = int.parse(args[2]);

        await firestore.FirebaseFirestore.instance
            .collection('MQTT')
            .doc('lightData')
            .update({
          'colors': "$red $green $blue",
          'isPhaseActive': 0,
          'time of change': firestore.Timestamp.now(),
        });
        await sendLightPhaseIsActiveToMQTT(false);

        await sendLightToMQTT(red, green, blue);

        print('Команда light успішно виконана: $red, $green,
$blue');
```

```
      } else if (commandName == 'lightPhase') {
        if (args.length != 7) {
          throw Exception('Неправильна кількість аргументів для
lightPhase');
        }

        final int r1 = int.parse(args[0]);
        final int g1 = int.parse(args[1]);
        final int b1 = int.parse(args[2]);
        final int r2 = int.parse(args[3]);
        final int g2 = int.parse(args[4]);
        final int b2 = int.parse(args[5]);
        final int seconds = int.parse(args[6]);

        await firestore.FirebaseFirestore.instance
            .collection('MQTT')
            .doc('lightData')
            .update({
          'phaseColors': "$r1 $g1 $b1 $r2 $g2 $b2 $seconds",
          'isPhaseActive': 1,
          'time of change': firestore.Timestamp.now(),
        });
        await sendLightPhaseIsActiveToMQTT(true);

        await sendLightPhaseToMQTT(r1, g1, b1, r2, g2, b2,
seconds);

        print(
            'Команда lightPhase успішно виконана: $r1, $g1, $b1 ->
$r2, $g2, $b2, $seconds сек');
      } else {
        print('Команда "$commandName" не підтримується.');
      }
    } catch (e) {
      print('Помилка при обробці команди: $e');
    }
  }

  Future<void> sendChatGPTAnswearPulseToMQTT(bool
chatGPTAnswerPulse) async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic =
'BerezaOleksandr/berezaLight/lightData/ChatGPTAnswerPulse';
    final payload = '$chatGPTAnswerPulse';
    await _loadSettings();

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      client.logging(on: true);
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;
```

```
      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();

        if (client.connectionStatus?.state ==
MqttConnectionState.connected) {
          print('Підключено до $brokerIp!');
          return true;
        } else {
          print('Стан підключення:
${client.connectionStatus?.state}');
        }
      } catch (e) {
        print('Помилка підключення до $brokerIp: $e');
      }
      return false;
    }

    final client = MqttServerClient(localBrokerIp, clientId);

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(
      topic,
      MqttQos.atLeastOnce,
      builder.payload!,
    );

    print('Дані успішно відправлено: $payload');

    client.disconnect();
  }

  Future<void> sendLightPhaseIsActiveToMQTT(bool isPhaseActive)
async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic =
'BerezaOleksandr/berezaLight/lightData/isPhaseActive';
    final payload = '$isPhaseActive';
    await _loadSettings();

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
```

```
      client.server = brokerIp;
      client.logging(on: true);
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;

      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();

        if (client.connectionStatus?.state ==
MqttConnectionState.connected) {
          print('Підключено до $brokerIp!');
          return true;
        } else {
          print('Стан підключення:
${client.connectionStatus?.state}');
        }
      } catch (e) {
        print('Помилка підключення до $brokerIp: $e');
      }
      return false;
    }

    final client = MqttServerClient(localBrokerIp, clientId);

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(
      topic,
      MqttQos.atLeastOnce,
      builder.payload!,
    );

    print('Дані успішно відправлено: $payload');

    client.disconnect();
  }

  Future<void> _loadSettings() async {
    final settings = await SettingsManager.loadSettings();
    setState(() {
      localBrokerIp = settings['localIp'];
      vpnBrokerIp = settings['vpnIp'];
      accentColor = settings['accentColor'];
```

```
      btnBgCol = accentColor;
    });
  }

  Future<void> sendLightPhaseToMQTT(
      int r1, int g1, int b1, int r2, int g2, int b2, int seconds)
async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic =
'BerezaOleksandr/berezaLight/lightData/phaseColors';
    final payload = '$r1 $g1 $b1 $r2 $g2 $b2 $seconds';
    await _loadSettings();

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      client.logging(on: true);
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;

      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();

        if (client.connectionStatus?.state ==
MqttConnectionState.connected) {
          print('Підключено до $brokerIp!');
          return true;
        } else {
          print('Стан підключення:
${client.connectionStatus?.state}');
        }
      } catch (e) {
        print('Помилка підключення до $brokerIp: $e');
      }
      return false;
    }

    final client = MqttServerClient(localBrokerIp, clientId);

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('Не вдалося підключитись до жодного брокера.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(payload);

    client.publishMessage(
      topic,
```

```
      MqttQos.atLeastOnce,
      builder.payload!,
    );

    print('Дані успішно відправлено: $payload');

    client.disconnect();
  }

  Future<void> sendLightPhaseToFirebaseAndMQTT(
      int r1, int g1, int b1, int r2, int g2, int b2, int seconds)
async {
    await firestore.FirebaseFirestore.instance
        .collection('MQTT')
        .doc('lightData')
        .update({
      'phaseColors': '$r1 $g1 $b1 $r2 $g2 $b2 $seconds',
      'isPhaseActive': 1,
      'time of change': firestore.Timestamp.now(),
    });
    await sendLightPhaseToMQTT(r1, g1, b1, r2, g2, b2, seconds);
  }

  Future<List<int>> _getAudioContent() async {
    final directory = await getApplicationDocumentsDirectory();
    final path = '${directory.path}/audio.wav';
    return File(path).readAsBytesSync().toList();
  }

  Future<void> sendToChatGPT() async {
    try {
      String chatResponse = await
getChatGPTResponse(recognizedTextFull);
      await saveResponseToFirebase(
          chatResponse);
      setState(() {
        List<String> splitResponse =
chatResponse.split('CHAT_COMMAND:');
        responseTextFull =
            splitResponse[0].replaceFirst('CHAT_TEXT:',
'').trim();
        responseCommandFull =
            splitResponse.length > 1 ? splitResponse[1].trim() :
'';
        if (responseCommandFull == '') {
          responseForUser = "$responseTextFull";
        } else {
          responseForUser =
              "$responseTextFull \n Команда:
$responseCommandFull";
        }
      });
```

```
    } catch (e) {
      setState(() {
        responseTextFull = 'Помилка: ${e.toString()}';
      });
    }
    if (_shouldSpeak) {
      speakText(responseTextFull, "uk-UA");
    }
  }

  Future<void> stopRecording() async {
    await _record.stop('my_recorder');
    setState(() {
      isRecording = false;
      recognizedText =
          'Обробляємо аудіо...';
      micColor = Colors.white60;
      btnBgCol = Colors.white12;
      icCol = 60;
    });

    try {
      final audio = await _getAudioContent();
      print('Аудіофайл завантажений: ${audio.length} байт');

      final response = await speech.recognize(config, audio);
      print('Отримано відповідь від Google API:
${response.results}');

      setState(() {
        recognizedTextFull = response.results
            .map((e) => e.alternatives.first.transcript)
            .join('\n');
        recognizedText = recognizedTextFull.isNotEmpty
            ? 'Розпізнано:'
            : 'Нічого не розпізнано. \n Спробуйте ще раз.';
        userInputHintText = recognizedTextFull;
      });
    } catch (e) {
      print('Помилка під час розпізнавання: $e');
      setState(() {
        recognizedText = 'Помилка при розпізнаванні.';
      });
    }
    try {
      final snapshot = await firestore.FirebaseFirestore.instance
          .collection('MQTTRequests')
          .orderBy('timestamp',
              descending: true)
          .limit(1)
          .get();
```

```
      int newId = 1;

      if (snapshot.docs.isNotEmpty) {
        String lastId = snapshot.docs.first.id;
        newId = int.parse(lastId) + 1;
      }

      await firestore.FirebaseFirestore.instance
          .collection('MQTTRequests')
          .doc(newId.toString())
          .set({
        'text': recognizedTextFull.isNotEmpty ? recognizedTextFull
: "Помилка",
        'timestamp': firestore.Timestamp.now(),
      });
      await sendToChatGPT();
      await updateCommandInFirestore(responseCommandFull);

      print("Нові дані додано успішно!");
      reqSendState = "Нові дані додано успішно!";
    } catch (e) {
      print("Помилка при зміні даних документа: $e");
      reqSendState = "Помилка при зміні даних документа.";
    }
    setState(() {
      micColor = Colors.white;
      btnBgCol = accentColor;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color.fromARGB(255, 10, 10, 10),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            SizedBox(height: 24),
            Expanded(
              child: Column(
                children: [
                  const SizedBox(height: 20),
                  Container(
                    padding: const EdgeInsets.all(16),
                    decoration: BoxDecoration(
                      border: Border.all(
                        color: Colors.white60,
                        width: 2.0,
                      ),
                      color: const Color.fromARGB(255, 18, 18,
18),
```

```
                    borderRadius: const
BorderRadius.all(Radius.circular(24)),
                  ),
                  constraints: const BoxConstraints(
                    maxHeight:
                        240,
                    maxWidth: 336,
                  ),
                  child: SingleChildScrollView(
                    child: Text(
                      responseForUser,
                      style: const TextStyle(
                        fontSize: 16,
                        color: Colors.white60,
                      ),
                      textAlign: TextAlign.center,
                    ),
                  ),
                ),
                Row(
                  mainAxisAlignment: MainAxisAlignment.end,
                  children: [
                    IconButton(
                      icon: Icon(
                        _shouldSpeak ? Icons.speaker :
Icons.speaker_outlined,
                        color: _shouldSpeak ? accentColor :
Colors.grey,
                        size: 30,
                      ),
                      onPressed: () {
                        setState(() {
                          _shouldSpeak = !_shouldSpeak;
                        });
                      },
                    ),
                    IconButton(
                      icon: Icon(
                        _shouldSpeak ? Icons.refresh :
Icons.refresh,
                        color: _shouldSpeak ? accentColor :
Colors.grey,
                        size: 40,
                      ),
                      onPressed: () {
                        replayAudioResponse();
                      },
                    ),
                  ],
                ),
              ],
            ),
```

```
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Container(
                  width: 286,
                  padding:
                      const EdgeInsets.symmetric(horizontal: 16,
vertical: 8),
                  decoration: BoxDecoration(
                    color: const Color.fromARGB(255, 18, 18, 18),
                    borderRadius: const
BorderRadius.all(Radius.circular(24)),
                    border: Border.all(
                      color: Colors.white60,
                      width: 2.0,
                    ),
                  ),
                  child: Row(
                    children: [
                      Flexible(
                        child: _isTextFieldActive
                            ? TextField(
                                controller: _textController,
                                decoration: const InputDecoration(
                                  hintText: 'Напишіть тут свій
запит..',

                                  hintStyle: TextStyle(
                                    color: Colors.white54,
                                    fontSize: 14,
                                  ),
                                  border: InputBorder.none,
                                ),
                                style: const TextStyle(
                                  color: Colors.white,
                                  fontSize: 16,
                                ),
                              )
                            : Container(
                                width: 202,
                                child: GestureDetector(
                                  onTap: () {
                                    setState(() {
                                      _isTextFieldActive = true;
                                    });
                                  },
                                  child: Text(
                                    userInputHintText,
                                    style: const TextStyle(
                                      color: Colors.white,
                                    ),
                                  ),
```

```
                                ),
                              ),
                        ),
                        IconButton(
                          icon: const Icon(Icons.send, color:
Colors.white),
                          onPressed: _sendText,
                        ),
                      ],
                    ),
                  ),
                  const SizedBox(
                      width: 8),
                  Container(
                    width: 50,
                    height: 50,
                    decoration: BoxDecoration(
                      border: Border.all(
                        color: micColor.withAlpha(40),
                        width: 2.5,
                      ),
                      color: accentColor,
                      borderRadius: const
BorderRadius.all(Radius.circular(50)),
                    ),
                    child: IconButton(
                      highlightColor: Colors.white24,
                      icon: Icon(
                        isRecording ? Icons.stop : Icons.mic,
                        size: 25,
                        color: micColor,
                      ),
                      onPressed: isRecording ? stopRecording :
startRecording,
                    ),
                  ),
                ],
              ),
              const SizedBox(height: 25),
            ],
          ),
        ),
      );
  }

  @override
  void dispose() {
    _record.dispose('my_recorder');
    audioPlayer.dispose();
    super.dispose();
  }
}
```

```
class AutomationPage extends StatefulWidget {
  const AutomationPage({super.key});

  @override
  State<AutomationPage> createState() => _AutomationPageState();
}

class _AutomationPageState extends State<AutomationPage> {
  String localBrokerIp = '192.168.0.106';
  String vpnBrokerIp = '192.168.0.144';
  List<Map<String, dynamic>> automations = [];

  @override
  void initState() {
    super.initState();
    _loadSettings();
    _connectAndLoadAutomations();
  }

  Future<void> _loadSettings() async {
    final settings = await SettingsManager.loadSettings();
    setState(() {
      localBrokerIp = settings['localIp'];
      vpnBrokerIp = settings['vpnIp'];
    });
  }

  Future<void> _connectAndLoadAutomations() async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic = 'Automation';
    await _loadSettings();

    final client = MqttServerClient(localBrokerIp, clientId);
    client.logging(on: true);
    client.keepAlivePeriod = 6;
    client.onDisconnected = _onDisconnected;
    client.onConnected = _onConnected;

    Future<bool> tryConnect(String brokerIp) async {
      client.server = brokerIp;
      try {
        print('🔌 Підключення до $brokerIp...');
        await client.connect();
        return client.connectionStatus?.state ==
MqttConnectionState.connected;
      } catch (e) {
        print('❌ Помилка підключення: $e');
        return false;
      }
    }
```

```
    if (!await tryConnect(localBrokerIp)) {
      if (!await tryConnect(vpnBrokerIp)) {
        print('✗ Не вдалося підключитись до жодного брокера');
        return;
      }
    }

    client.subscribe(topic, MqttQos.atLeastOnce);

    client.updates!.listen((List<MqttReceivedMessage<MqttMessage>>
c) {
      final recMess = c[0].payload as MqttPublishMessage;
      final payloadStr =

MqttPublishPayload.bytesToStringAsString(recMess.payload.message);

      try {
        final decoded = jsonDecode(payloadStr);
        if (decoded is Map<String, dynamic> &&
            decoded.containsKey('automations')) {
          final loadedList =
              List<Map<String,
dynamic>>.from(decoded['automations']);
          setState(() {
            automations = loadedList;
          });
          print('✓ Завантажено автоматизацій:
${automations.length}');
        }
      } catch (e) {
        print('✗ Неможливо розпарсити JSON: $payloadStr');
      }
    });
  }

  void _onConnected() => print('✓ Підключено до MQTT');
  void _onDisconnected() => print('🔌 Відключено від MQTT');

  void _sendFullAutomationsListToMQTT() async {
    const clientId = 'BerezaOleksandr_PhoneAppClient';
    const topic = 'Automation';
    await _loadSettings();

    final fullJson = jsonEncode({'automations': automations});
    final client = MqttServerClient(localBrokerIp, clientId);

    Future<bool> tryConnect(MqttServerClient client, String
brokerIp) async {
      client.server = brokerIp;
      client.logging(on: true);
```

```
      client.keepAlivePeriod = 6;
      client.onDisconnected = _onDisconnected;

      try {
        print('Спроба підключитись до $brokerIp...');
        await client.connect();
        return client.connectionStatus?.state ==
MqttConnectionState.connected;
      } catch (e) {
        print('✗ Помилка MQTT-підключення: $e');
        return false;
      }
    }

    if (!await tryConnect(client, localBrokerIp)) {
      if (!await tryConnect(client, vpnBrokerIp)) {
        print('✗ Обидва брокери недоступні.');
        return;
      }
    }

    final builder = MqttClientPayloadBuilder();
    builder.addString(fullJson);

    client.publishMessage(topic, MqttQos.atLeastOnce,
builder.payload!,
        retain: true);

    print('⬆ Відправлено весь список автоматизацій');

    client.disconnect();
  }

  void _showAddAutomationTypeDialog() {
    showModalBottomSheet(
      context: context,
      builder: (context) => Wrap(
        children: [
          ListTile(
            leading: const Icon(Icons.access_time),
            title: const Text("За часом"),
            onTap: () {
              Navigator.pop(context);
              _openTimeAutomationDialog();
            },
          ),
          ListTile(
            leading: const Icon(Icons.rule),
            title: const Text("За умовою"),
            onTap: () {
              Navigator.pop(context);
```

```
                _openConditionalAutomationDialog();
              },
            ),
          ],
        ),
      );
    }

    void _openConditionalAutomationDialog() {
      String? selectedParameter;
      String selectedOperator = '>';
      String inputValue = '';
      String? selectedCommand;
      String args = "";

      showDialog(
        context: context,
        builder: (context) {
          return StatefulBuilder(builder: (context, setState) {
            bool isValid = selectedParameter != null &&
                inputValue.trim().isNotEmpty &&
                selectedCommand != null &&
                args.trim().isNotEmpty;

            return AlertDialog(
              title: const Text('Нова автоматизація (за умовою)'),
              content: SingleChildScrollView(
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    DropdownButton<String>(
                      value: selectedParameter,
                      hint: const Text("Параметр"),
                      isExpanded: true,
                      items: const [
                        DropdownMenuItem(
                            value: 'temperature', child:
Text('Температура')),
                        DropdownMenuItem(
                            value: 'humidity', child:
Text('Вологість')),
                        DropdownMenuItem(
                            value: 'light', child:
Text('Освітленість')),
                        DropdownMenuItem(
                            value: 'mq9', child: Text('Шкідливі
гази')),
                      ],
                      onChanged: (value) {
                        setState(() => selectedParameter = value);
                      },
                    ),
```

```
                    DropdownButton<String>(
                      value: selectedOperator,
                      isExpanded: true,
                      items: const [
                        DropdownMenuItem(value: '>', child:
Text('Більше')),
                        DropdownMenuItem(value: '<', child:
Text('Менше')),
                        DropdownMenuItem(
                            value: '>=', child: Text('Більше або
рівне')),
                        DropdownMenuItem(
                            value: '<=', child: Text('Менше або
рівне')),
                        DropdownMenuItem(value: '==', child:
Text('Дорівнює')),
                        DropdownMenuItem(value: '!=', child:
Text('Не дорівнює')),
                      ],
                      onChanged: (value) {
                        setState(() => selectedOperator = value!);
                      },
                    ),
                    TextField(
                      decoration: const InputDecoration(labelText:
"Значення"),
                      keyboardType: TextInputType.number,
                      onChanged: (value) => setState(() =>
inputValue = value),
                    ),
                    const Divider(height: 20),
                    DropdownButton<String>(
                      value: selectedCommand,
                      hint: const Text("Команда"),
                      isExpanded: true,
                      items: const [
                        DropdownMenuItem(
                            value: 'light', child: Text('Звичайне
світло')),
                        DropdownMenuItem(
                            value: 'lightPhase',
                            child: Text('Переливання світла')),
                      ],
                      onChanged: (value) {
                        setState(() => selectedCommand = value);
                      },
                    ),
                    TextField(
                      decoration:
                          const InputDecoration(labelText:
"Аргументи команди"),
```

```
                    onChanged: (value) => setState(() => args =
value),
                  ),
                ],
              ),
            ),
            actions: [
              TextButton(
                onPressed: () => Navigator.pop(context),
                child: const Text("Скасувати"),
              ),
              ElevatedButton(
                onPressed: isValid
                    ? () {
                        final automation = {
                          "type": "condition",
                          "parameter": selectedParameter,
                          "operator": selectedOperator,
                          "value": inputValue.trim(),
                          "command": selectedCommand,
                          "args": args.trim(),
                        };
                        setState(() {
                          automations.add(automation);
                        });
                        _sendFullAutomationsListToMQTT();
                        Navigator.pop(context);
                      }
                    : null,
                child: const Text("Зберегти"),
              ),
            ],
          );
        });
      },
    );
  }

  void _openTimeAutomationDialog() {
    String? selectedCommand;
    String args = "";
    DateTime? selectedDateTime;
    bool isRepeating = false;

    showDialog(
      context: context,
      builder: (context) {
        return StatefulBuilder(builder: (context, setState) {
          bool isValid = selectedCommand != null &&
              args.trim().isNotEmpty &&
              selectedDateTime != null;
```

```
          return AlertDialog(
            title: const Text('Нова автоматизація (за часом)'),
            content: SingleChildScrollView(
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  DropdownButton<String>(
                    value: selectedCommand,
                    hint: const Text("Команда"),
                    isExpanded: true,
                    items: const [
                      DropdownMenuItem(
                          value: 'light', child: Text('Звичайне
світло')),
                      DropdownMenuItem(
                          value: 'lightPhase',
                          child: Text('Переливання світла')),
                    ],
                    onChanged: (value) {
                      setState(() => selectedCommand = value);
                    },
                  ),
                  TextField(
                    decoration:
                        const InputDecoration(labelText:
"Аргументи команди"),
                    onChanged: (value) => setState(() => args =
value),
                  ),
                  const SizedBox(height: 10),
                  ListTile(
                    title: const Text("Дата та час"),
                    subtitle: Text(
                      selectedDateTime != null
                          ? DateFormat('dd.MM.yyyy HH:mm')
                              .format(selectedDateTime!)
                          : "Не вибрано",
                    ),
                    onTap: () async {
                      final pickedDate = await showDatePicker(
                        context: context,
                        initialDate: DateTime.now(),
                        firstDate: DateTime.now(),
                        lastDate: DateTime(2030),
                      );
                      if (pickedDate != null) {
                        final pickedTime = await showTimePicker(
                          context: context,
                          initialTime: TimeOfDay.now(),
                        );
                        if (pickedTime != null) {
                          setState(() {
```

```
                                selectedDateTime = DateTime(
                                  pickedDate.year,
                                  pickedDate.month,
                                  pickedDate.day,
                                  pickedTime.hour,
                                  pickedTime.minute,
                                );
                              });
                            }
                          }
                        },
                      ),
                      SwitchListTile(
                        title: const Text("Повторювати"),
                        value: isRepeating,
                        onChanged: (val) => setState(() => isRepeating
= val),
                      ),
                    ],
                  ),
                ),
                actions: [
                  TextButton(
                    onPressed: () => Navigator.pop(context),
                    child: const Text("Скасувати"),
                  ),
                  ElevatedButton(
                    onPressed: isValid
                        ? () {
                            final automation = {
                              "type": "time",
                              "command": selectedCommand,
                              "args": args.trim(),
                              "datetime": DateFormat("dd.MM.yyyy
HH:mm")
                                  .format(selectedDateTime!),
                              "repeat": isRepeating
                            };

                            setState(() {
                              automations.add(automation);
                            });
                            _sendFullAutomationsListToMQTT();
                            Navigator.pop(context);
                          }
                        : null,
                    child: const Text("Зберегти"),
                  ),
                ],
              );
            });
          },
```

```
    );
  }

  void _editAutomationDialog(int index) {
    final automation = automations[index];
    String selectedCommand = automation["command"];
    String args = automation["args"];
    DateTime selectedDateTime =
        DateFormat("dd.MM.yyyy
HH:mm").parse(automation["datetime"]);
    bool isRepeating =
        automation["repeat"] == true || automation["repeat"] ==
"1";

    showDialog(
      context: context,
      builder: (context) {
        return StatefulBuilder(builder: (context, setState) {
          bool isValid = selectedCommand.isNotEmpty &&
              args.trim().isNotEmpty &&
              selectedDateTime != null;

          return AlertDialog(
            title: const Text('Редагування автоматизації'),
            content: SingleChildScrollView(
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  DropdownButton<String>(
                    value: selectedCommand,
                    hint: const Text("Команда"),
                    isExpanded: true,
                    items: const [
                      DropdownMenuItem(
                          value: 'light', child: Text('Звичайне
світло')),
                      DropdownMenuItem(
                          value: 'lightPhase',
                          child: Text('Переливання світла')),
                    ],
                    onChanged: (value) {
                      if (value != null) {
                        setState(() => selectedCommand = value);
                      }
                    },
                  ),
                  TextField(
                    controller: TextEditingController(text: args),
                    decoration:
                        const InputDecoration(labelText:
"Аргументи команди"),
```

```
                    onChanged: (value) => setState(() => args =
value),
                ),
                const SizedBox(height: 10),
                ListTile(
                  title: const Text("Дата та час"),
                  subtitle: Text(
                    DateFormat('dd.MM.yyyy
HH:mm').format(selectedDateTime),
                  ),
                  onTap: () async {
                    final pickedDate = await showDatePicker(
                      context: context,
                      initialDate: selectedDateTime,
                      firstDate: DateTime.now(),
                      lastDate: DateTime(2030),
                    );
                    if (pickedDate != null) {
                      final pickedTime = await showTimePicker(
                        context: context,
                        initialTime:
TimeOfDay.fromDateTime(selectedDateTime),
                      );
                      if (pickedTime != null) {
                        setState(() {
                          selectedDateTime = DateTime(
                            pickedDate.year,
                            pickedDate.month,
                            pickedDate.day,
                            pickedTime.hour,
                            pickedTime.minute,
                          );
                        });
                      }
                    }
                  },
                ),
                SwitchListTile(
                  title: const Text("Повторювати"),
                  value: isRepeating,
                  onChanged: (val) => setState(() => isRepeating
= val),
                ),
              ],
            ),
          ),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: const Text("Скасувати"),
            ),
            ElevatedButton(
```

```
                onPressed: isValid
                    ? () {
                        final updatedAutomation = {
                          "type": "time",
                          "command": selectedCommand,
                          "args": args.trim(),
                          "datetime": DateFormat("dd.MM.yyyy
HH:mm")
                              .format(selectedDateTime),
                          "repeat": isRepeating
                        };

                        setState(() {
                          automations[index] = updatedAutomation;
                        });
                        _sendFullAutomationsListToMQTT();
                        Navigator.pop(context);
                      }
                    : null,
                child: const Text("Зберегти"),
              ),
            ],
          );
        });
      },
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Автоматизації')),
      body: ListView.builder(
        itemCount: automations.length,
        itemBuilder: (context, index) {
          final automation = automations[index];
          return Dismissible(
            key: Key(automation['datetime'].toString() +
automation['command']),
            direction: DismissDirection.endToStart,
            background: Container(
              color: Colors.red,
              alignment: Alignment.centerRight,
              padding: const EdgeInsets.symmetric(horizontal: 20),
              child: const Icon(Icons.delete, color:
Colors.white),
            ),
            confirmDismiss: (_) async {
              return await showDialog(
                context: context,
                builder: (_) => AlertDialog(
                  title: const Text("Видалити автоматизацію?"),
```

```
                    content: const Text("Цю дію не можна буде
скасувати."),
                    actions: [
                      TextButton(
                          onPressed: () => Navigator.pop(context,
false),
                          child: const Text("Скасувати")),
                      TextButton(
                          onPressed: () => Navigator.pop(context,
true),
                          child: const Text("Видалити")),
                    ],
                  ),
                );
              },
              onDismissed: (direction) {
                setState(() => automations.removeAt(index));
                _sendFullAutomationsListToMQTT();
              },
              child: ListTile(
                leading: const Icon(Icons.flash_on),
                title: Text('${automation["command"]}
(${automation["args"]})'),
                subtitle: Text(
                  'Дата: ${automation["datetime"]} | Повторювати:
${automation["repeat"]}',
                ),
                trailing: IconButton(
                  icon: const Icon(Icons.edit, color:
Colors.orange),
                  onPressed: () => _editAutomationDialog(index),
                ),
              ),
            );
          },
        ),
        floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.green,
          onPressed: _showAddAutomationTypeDialog,
          child: const Icon(Icons.add),
        ),
      );
    }
}

class FirebaseScreen extends StatefulWidget {
  const FirebaseScreen({Key? key}) : super(key: key);

  @override
  State<FirebaseScreen> createState() => _FirebaseScreenState();
}
```

```
class _FirebaseScreenState extends State<FirebaseScreen> {
  String? _selectedCollection;
  List<Map<String, dynamic>> _documents = [];

  final List<String> _collections = ['MQTTRequests',
'MQTTResponses', 'MQTT'];

  Future<void> _loadDocuments(String collection) async {
    final snapshot = await firestore.FirebaseFirestore.instance
        .collection(collection)
        .orderBy('timestamp', descending: true)
        .get();

    setState(() {
      _selectedCollection = collection;
      _documents = snapshot.docs.map((doc) {
        Map<String, dynamic> data = doc.data();

        if (data['timestamp'] is firestore.Timestamp) {
          data['timestamp'] =
              (data['timestamp'] as firestore.Timestamp).toDate();
        }
        return data;
      }).toList();
    });

    if (collection == 'MQTT') {
      _documents.clear();

      final lightDoc = await firestore.FirebaseFirestore.instance
          .collection('MQTT')
          .doc('lightData')
          .get();
      if (lightDoc.exists) {
        setState(() {
          _documents.add({'id': 'lightData',
...lightDoc.data()!});
        });
      }
      final SensorDataDoc = await
firestore.FirebaseFirestore.instance
          .collection('MQTT')
          .doc('SensorData')
          .get();
      if (SensorDataDoc.exists) {
        setState(() {
          _documents.add({'id': 'SensorData',
...SensorDataDoc.data()!});
        });
      }
    }
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color.fromARGB(255, 10, 10, 10),
      body: Row(
        children: [
          Expanded(
            child: Container(
              color: const Color.fromARGB(255, 14, 14, 14),
              child: ListView.builder(
                itemCount: _collections.length,
                itemBuilder: (context, index) {
                  return ListTile(
                    title: Text(_collections[index]),
                    titleTextStyle: const TextStyle(color:
Colors.white),
                    onTap: () =>
_loadDocuments(_collections[index]),
                  );
                },
              ),
            ),
          ),
          Container(
            color: const Color.fromARGB(255, 10, 10, 10),
            width: 200,
            child: _selectedCollection == null
                ? const Center(
                    child: Text(
                    'Виберіть колекцію',
                    style: TextStyle(color: Colors.white),
                  ))
                : ListView.builder(
                    itemCount: _documents.length,
                    itemBuilder: (context, index) {
                      final document = _documents[index];
                      String colTitle = 'Без тексту';
                      String colSubtitle = 'Без тексту';
                      String formattedDate = document['timestamp']
!= null
                          ? DateFormat('dd-MM-yyyy HH:mm:ss')
                              .format(document['timestamp'])
                          : 'Немає дати';

                      if (_selectedCollection == 'MQTTRequests') {
                        colTitle = document['text'] ?? 'Без
тексту';
                        colSubtitle = formattedDate;
                      } else if (_selectedCollection ==
'MQTTResponses') {
```

```
                        colTitle = document['chat_text'] ?? 'Без
тексту';
                        colSubtitle =
                            "Команда:
${document['chat_command']}\n Час:
${document['timestamp']?.toString()}";
                      } else if (_selectedCollection == 'MQTT') {
                        if (document['id'] == 'lightData') {
                          colTitle = 'Light Data';
                          colSubtitle =
                              'RGB Colors: ${document['colors']}\n
Is phase: ${document['isPhaseActive']}\n Phase RGB:\n
${document['phaseColors']}';
                        } else if (document['id'] == 'SensorData')
{
                          colTitle = 'Sensor Data';
                          colSubtitle =
                              'Вн. температура:
${document['indoorTemperature']}\n Вн. вологість:
${document['indoorHumidity']}\n Зовн. температура:
${document['outdoorTemperature']}\n Зовн. вологість:
${document['outdoorHumidity']}\n Швидкість вітру:
${document['windSpeed']}\n Напрямок вітру:
${document['windDegree']}\n Опис:
${document['weatherDescription']}';
                        }
                      }

                      return Card(
                        color: const Color.fromARGB(255, 18, 18,
18),
                        child: ListTile(
                          tileColor: const Color.fromARGB(255, 18,
18, 18),
                          title: Text(colTitle,
                              style: const TextStyle(color:
Colors.white)),
                          subtitle: Text(colSubtitle,
                              style: const TextStyle(color:
Colors.white60)),
                        ),
                      );
                    },
                  ),
              ),
            ],
          ),
        );
    }
}

class SettingsManager {
```

```
  static Future<void> saveSettings(
      String localIp, String vpnIp, Color accentColor) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('localIp', localIp);
    await prefs.setString('vpnIp', vpnIp);
    await prefs.setInt('accentColor', accentColor.value);
  }

  static Future<Map<String, dynamic>> loadSettings() async {
    final prefs = await SharedPreferences.getInstance();
    return {
      'localIp': prefs.getString('localIp') ?? '192.168.0.106',
      'vpnIp': prefs.getString('vpnIp') ?? '192.168.195.144',
      'accentColor': Color(prefs.getInt('accentColor') ??
0xFF807FFC),
    };
  }

  static Future<void> saveCustomInstructions(String text) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('customInstructions', text);
  }

  static Future<String> loadCustomInstructions() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('customInstructions') ?? '';
  }
}

class PresetManager {
  static Future<void> savePreset(
      String name, int red, int green, int blue) async {
    final prefs = await SharedPreferences.getInstance();
    List<String> presets = prefs.getStringList('presets') ?? [];
    presets.add('$name:$red,$green,$blue');
    await prefs.setStringList('presets', presets);
  }

  static Future<List<Map<String, dynamic>>> loadPresets() async {
    final prefs = await SharedPreferences.getInstance();
    List<String> presets = prefs.getStringList('presets') ?? [];
    return presets.map((preset) {
      var parts = preset.split(':');
      var colors = parts[1].split(',').map(int.parse).toList();
      return {
        'name': parts[0],
        'red': colors[0],
        'green': colors[1],
        'blue': colors[2]
      };
    }).toList();
  }
```

```
  static Future<void> deletePreset(String name) async {
    final prefs = await SharedPreferences.getInstance();
    List<String> presets = prefs.getStringList('presets') ?? [];
    presets.removeWhere((preset) => preset.startsWith(name));
    await prefs.setStringList('presets', presets);
  }
}

class SettingsScreen extends StatefulWidget {
  const SettingsScreen({super.key});

  @override
  _SettingsScreenState createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  late TextEditingController localIpController;
  late TextEditingController vpnIpController;
  TextEditingController _instructionController =
TextEditingController();

  Color accentColor = purpleAccentColor;

  @override
  void initState() {
    super.initState();
    _loadSettings();
    SettingsManager.loadCustomInstructions().then((value) {
      setState(() {
        _instructionController.text = value;
      });
    });
  }

  Future<void> _loadSettings() async {
    var settings = await SettingsManager.loadSettings();
    setState(() {
      localIpController = TextEditingController(text:
settings['localIp']);
      vpnIpController = TextEditingController(text:
settings['vpnIp']);
      accentColor = settings['accentColor'];
    });
  }

  void _saveSettings() {
    SettingsManager.saveSettings(
        localIpController.text, vpnIpController.text,
accentColor);
  }
```

```
  Widget _buildColorSlider(
      String label, int value, ValueChanged<int> onChanged) {
    return Column(
      children: [
        Text(label, style: const TextStyle(color: Colors.white)),
        Slider(
          value: value.toDouble(),
          min: 0,
          max: 255,
          divisions: 255,
          activeColor: accentColor,
          inactiveColor: Colors.grey,
          onChanged: (newValue) => onChanged(newValue.toInt()),
        ),
      ],
    );
  }

  void showColorPickerDialog(
      Color pickerColor, Function(Color) onColorSelected) {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        int tempRed = pickerColor.red;
        int tempGreen = pickerColor.green;
        int tempBlue = pickerColor.blue;

        return StatefulBuilder(
          builder: (context, setState) {
            return AlertDialog(
              backgroundColor: const Color.fromARGB(255, 30, 30,
30),
              title: const Text(
                'Вибір кольору',
                style: TextStyle(color: Colors.white),
              ),
              content: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  Container(
                    width: 80,
                    height: 80,
                    decoration: BoxDecoration(
                      color: Color.fromARGB(255, tempRed,
tempGreen, tempBlue),
                      shape: BoxShape.circle,
                      border: Border.all(color: Colors.white30,
width: 2),
                    ),
                  ),
                  const SizedBox(height: 10),
                  _buildColorSlider("Red", tempRed, (value) {
```

```
                    setState(() => tempRed = value);
                  }),
                  _buildColorSlider("Green", tempGreen, (value) {
                    setState(() => tempGreen = value);
                  }),
                  _buildColorSlider("Blue", tempBlue, (value) {
                    setState(() => tempBlue = value);
                  }),
                ],
              ),
              actions: [
                TextButton(
                  onPressed: () => Navigator.pop(context),
                  child: const Text("Скасувати",
                      style: TextStyle(color: Colors.red)),
                ),
                TextButton(
                  onPressed: () {
                    Color selectedColor =
                        Color.fromARGB(255, tempRed, tempGreen,
tempBlue);
                    onColorSelected(selectedColor);
                    Navigator.pop(context);
                  },
                  child: const Text("Зберегти",
                      style: TextStyle(color: Colors.green)),
                ),
              ],
            );
          },
        );
      },
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Налаштування')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextField(
              controller: localIpController,
              decoration:
                  const InputDecoration(labelText: 'Локальний IP
брокера'),
            ),
            TextField(
              controller: vpnIpController,
```

```
              decoration: const InputDecoration(labelText: 'VPN IP
брокера'),
            ),
            const SizedBox(height: 16),
            ListTile(
              title: const Text('Акцентний колір'),
              trailing: CircleAvatar(backgroundColor:
accentColor),
              onTap: () {
                showColorPickerDialog(accentColor, (newColor) {
                  setState(() {
                    accentColor = newColor;
                  });

SettingsManager.saveSettings(localIpController.text,
                      vpnIpController.text, accentColor);
                });
              },
            ),
            const SizedBox(height: 20),
            TextField(
              controller: _instructionController,
              decoration: const InputDecoration(
                labelText: 'Додаткові інструкції для ChatGPT',
                border: OutlineInputBorder(),
              ),
              maxLines: 5,
            ),
            const SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                _saveSettings();
                SettingsManager.saveCustomInstructions(
                    _instructionController.text);
              },
              child:
                  const Text('Зберегти', style: TextStyle(color:
Colors.green)),
            ),
          ],
        ),
      ),
    );
  }
}
```