Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКАКВАЛІФІКАЦІЙНА РОБОТА

на тему:

Комп'ютерна система керування IoT пристроями
08-54.БКР.010.00.000 ПЗ

Виконав: студент 4 курсу, групи 1КІ-23мс
спеціальності 123— «Комп'ютерна інженерія»
освітня програма – «Комп'ютерна інженерія»
Куляс О.П.

Керівник к.т.н., доц. каф. ОТ
Колесник І.С.

Рецензент к.т.н., зав. каф. МБІС
Карпінець В.В..

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

"____" _____ 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо—кваліфікаційний рівень бакалвр
Галузь знань — 12 Інформаційні технології
Спеціальність— 123 Комп'ютерна інженерія
Освітня програма— Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ проф., д.т.н.
_____Азаров О. Д.
"____"_____ 2025 року

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

Кулясу Олександру Павловичу

1 Тема роботи: Комп'ютерна система керування IoT пристроями, керівник роботи Колесник Ірина Сергіївна, к.т.н., доцент кафедри ОТ, затверджено наказом №97 вищого навчального закладу від « 20 » березня 2025 року.

2 Строк подання студентом роботи 13.05.25

3 Вихідні дані до роботи: опис існуючих комп'ютерних систем керування IoT пристроями, опис архітектури Bluetooth-зв'язку, мікроконтролер ESP32, мова програмування контролера C++, середовище Arduino IDE, мережа Wi-Fi, .

4 Зміст розрахунково-пояснювальної записки: вступ, огляд та аналіз галузі дослідження, теорстичні аспекти розробки системи, практична реалізація системи, висновки.

5 Перелік графічного матеріалу: Процес ініціалізації IoT пристрою, Послідовність взаємодії між мобільним клієнтом, MQTT брокером та IoT-пристроєм, принцип роботи IoT пристрою.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к.т.н., доц. каф. ОТ Колесник І.С.	21.03.2025р.	13.06.2025р.
Нормоконтроль	ас.каф.ОТ Швець С.І.	14.05.2025р.	30.05.2025р.

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план приведений в таблиці 2.

Таблиця 2— Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Постановка задачі роботи	21.03.25	
2	Огляд та аналіз галузі дослідження,	21.03.25— 29.03.25	
3	Теоретичні аспекти розробки системи,	21.03.25— 29.03.25	
4	Практична реалізація системи	30.03.25— 11.04.25	
5	Реалізація апаратної частини IoT-рішення	12.04.25— 26.04.25	
6	Розробка програмного забезпечення пристрою	27.04.25— 07.05.25	
7	Оформлення пояснювальної записки та ілюстративного матеріалу	08.05.25	
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	14.05.25	

Студент Олександр КУЛЯС

Керівник роботи к.т.н., доц. Ірина КОЛЕСНИК

АНОТАЦІЯ

УДК 681.518

Куляс О.П. Комп'ютерна система керування IoT пристроями.

На укр. мові. Бібліогр. 26 назв; рис. : 22; табл.: 3; аркушів.: 72.

Кваліфікаційна робота зосереджена розробці комп'ютеризованої системи керування IoT-пристроями для "розумного будинку", що функціонує на основі розпізнавання жестів, зафіксованих камерою смартфона з операційною системою Android. Також у межах проєкту реалізовано власний IoT-пристрій, адаптований для взаємодії з цією системою.

У ході роботи проаналізовано наявні технічні рішення, що стосуються систем управління жестами, а також досліджено інструменти для обробки відеопотоку з метою виявлення жестових команд. Розглянуто принципи побудови комунікації між клієнтським додатком та IoT-пристроями, спроектовано архітектуру всієї системи, включаючи Android-додаток та апаратну складову інтелектуального пристрою.

Розроблено Android-клієнт із графічним інтерфейсом користувача, функцією розпізнавання жестів у реальному часі, а також засобами керування підключеним IoT-пристроєм. В межах проєкту створено прототип "розумної" розетки, включно з її апаратною та програмною реалізацією. Описано структуру системи з акцентом на її ключові компоненти та взаємодію між ними.

Проведено експериментальну перевірку працездатності створеного рішення, що дозволило оцінити його ефективність. Результати свідчать про практичну доцільність та можливість подальшого використання системи для автоматизації керування пристроями в умовах "розумного будинку".

Ключові слова: відеопотік, розпізнавання жестів, інтелектуальна розетка, Інтернет речей (IoT), Arduino, Android

ABSTRACT

Kulyas O.P. Computer System for IoT Device Management.

In Ukrainian. Bibliography: 26 titles; fig.: 22; tab.: 3; sheets: 72.

The bachelor's qualification work is dedicated to the development of a computerized system for managing IoT devices within a smart home environment. The system operates based on hand gesture recognition captured by the camera of an Android smartphone. Additionally, a custom IoT device was developed as part of the project, adapted for integration with the system.

The study includes an analysis of existing technical solutions related to gesture control systems, as well as an examination of software tools for processing video streams to detect gesture commands. The principles of interaction between the client application and IoT devices are considered, and the architecture of the entire system is designed, including both the Android application and the hardware component of the intelligent device.

An Android client was developed with a user-friendly interface, real-time gesture recognition functionality, and tools for controlling the connected IoT device. As part of the project, a prototype of a smart socket was created, including both hardware and software implementation. The system structure is described, highlighting its main components and their interaction.

Experimental testing of the developed solution was carried out to evaluate its performance. The results demonstrate its effectiveness and the potential for further application in automating device control in a smart home setting.

Keywords: video stream, gesture recognition, smart socket, Internet of Things (IoT), Arduino, Android

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі інтенсивний розвиток цифрових технологій сприяє впровадженню інтелектуальних систем в різні сфери людської діяльності. Однією з ключових складових такого розвитку є концепція Інтернету речей (IoT), яка передбачає інтеграцію пристроїв у мережу для обміну даними та автоматичного управління. Завдяки IoT з'являється можливість створення систем керування освітленням, температурою, безпекою, енергоспоживанням тощо, що суттєво підвищує комфорт і ефективність використання ресурсів.

Проте в процесі керування IoT-пристроями виникають певні труднощі, зокрема, необхідність використання декількох додатків від різних виробників, обмежені можливості взаємодії, а також закриті API. Це знижує гнучкість та ускладнює інтеграцію пристроїв у єдину систему. У зв'язку з цим актуальним є створення універсального рішення, яке б забезпечувало зручне, розширюване та централізоване управління IoT-пристроями.

Перспективним підходом є розробка комп'ютерної системи, що забезпечує взаємодію між користувачем та IoT-пристроями через зручний інтерфейс, зокрема, використовуючи можливості смартфонів, комп'ютерного зору та алгоритмів машинного навчання. Така система дозволить контролювати пристрої розумного будинку або офісу за допомогою інтуїтивно зрозумілих команд, включно з жестами.

Запропоноване рішення передбачає розробку прототипу програмно-апаратного комплексу, в якому користувач за допомогою смартфона з ОС Android зможе керувати IoT-пристроями. Це дозволить обійти обмеження, пов'язані з використанням сторонніх платформ, та забезпечить високу гнучкість для розширення системи.

Мета роботи — розробка комп'ютерної системи для централізованого управління IoT-пристроями із застосуванням жестового управління за

допомогою смартфона з ОС Android та створення власного IoT-пристрою для інтеграції в систему.

Завдання дослідження:

– провести аналіз існуючих рішень у сфері керування IoT-пристроями;

– дослідити можливості реалізації інтерфейсів керування на базі Android;

– розробити розширювану архітектуру системи для інтеграції нових пристроїв;

– реалізувати Android-клієнт з підтримкою керування жестами;

– створити власний IoT-пристрій із відкритим інтерфейсом керування;

– провести тестування ефективності та стабільності функціонування системи.

Об'єкт дослідження — система керування пристроями Інтернету речей.

Предмет дослідження — програмно-апаратні засоби для взаємодії з IoT-пристроями, зокрема на основі Android-платформи.

Методи дослідження. У ході виконання роботи використовувалися методи порівняльного аналізу, системного проєктування, технічного моделювання, а також експериментального тестування створеного рішення.

Практичне значення. Результатом роботи є прототип комп'ютерної системи для керування IoT-пристроями, що може стати основою для створення персоналізованих рішень у сфері автоматизації. Система вирізняється адаптивністю, можливістю масштабування та відкритістю для інтеграції нових функцій і пристроїв, що робить її придатною як для побутового, так і для промислового застосування.

1 ОГЛЯД ТА АНАЛІЗ ГАЛУЗІ ДОСЛІДЖЕННЯ

1.1 Огляд існуючих підходів до реалізації подібних систем

У процесі аналізу наявних рішень, що реалізують функціонал, аналогічний до цілей нашої розробки, було виявлено низку систем, які спрямовані на управління пристроями Інтернету речей.

Однією з найбільш схожих до нашої концепції є система Swipe від компанії Fibaro [1]. Вона підтримує широкий спектр IoT-пристроїв, зокрема як фірмових аксесуарів із закритим API, так і сторонніх пристроїв завдяки інтеграції з такими виробниками, як Philips та D-Link. Окрім цього, Swipe взаємодіє з екосистемами SmartThings (Samsung) та HomeKit (Apple), що, у свою чергу, забезпечують ще ширший спектр сумісного обладнання.

Користувацький інтерфейс представлений планшетом із вбудованою камерою, через який здійснюється управління пристроями. Основний елемент цієї системи показано на рис. 1.1.

Рисунок 1.1 — Вигляд системи Swipe

Як зазначено на офіційному сайті виробника [1], в межах одного житлового простору можливе використання лише одного керуючого планшета,

при цьому його необхідно розміщувати на горизонтальній поверхні. Це означає, що для взаємодії з системою користувач має фізично наблизитися до пристрою.

Ще одним прикладом є система WePe від компанії Maxustech, у якій управління відбувається через ультразвуковий сенсор, представлений на рис.1.2.

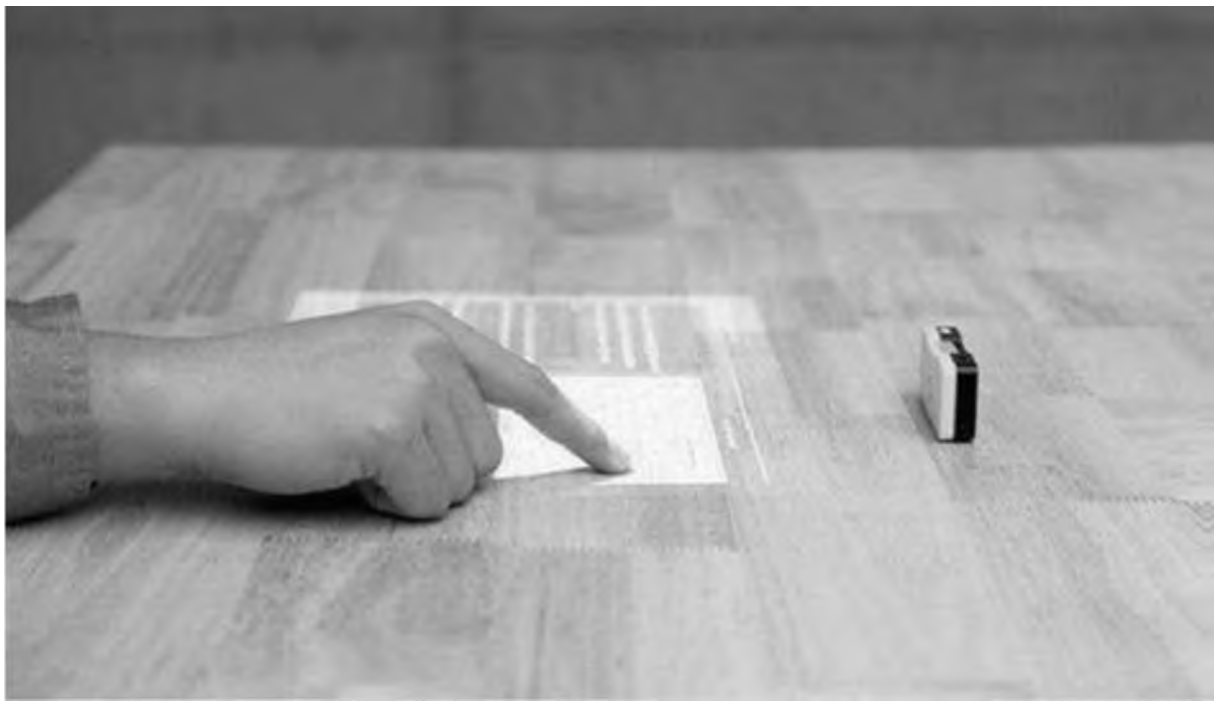

Рисунок 1.2 — Вигляд системи WePe

Як і в попередньому випадку, користувачу необхідно перебувати у безпосередній близькості до пристрою для коректної взаємодії.

Інша технологія, що заслуговує уваги, — Leap Motion. Це компактний пристрій, що підключається через USB-порт (рис. 1.3), який за допомогою інфрачервоного випромінювання формує тривимірну модель простору перед собою. Згідно з інформацією з офіційного джерела [2], зона розпізнавання жестів становить приблизно 227 дм³.

Ще одне цікаве рішення — Nod Gesture Control, розроблене компанією Nod. Пристрій реалізований у формі кільця (рис. 1.4), яке надягається на палець і дозволяє керувати смарт-пристроями за допомогою жестів. Система оснащена акселерометром і гіроскопом для точного відстеження рухів руки.

Рисунок 1.3 — Система Leap Motion

Рисунок 1.4 — Кільце Nod Gesture Control

На основі вивчених аналогів було сформовано порівняльну таблицю (табл. 1.1), в якій наведено ключові характеристики кожної з систем, а також переваги запропонованого рішення.

Таблиця 1.1 – Порівняння аналогів із пропонованим рішенням

	Swipe	Welle	Leap Motion	Nod Gesture Control	Пропоноване рішення
Безконтактне управління	+	+	+	+	+
Управління за допомогою різних девайсів	-	-	-	-	+
Відкритий API своїх IoT пристроїв	-	-	-	-	+
Широке охоплення території елементами керування системи	-	-	+	+	+
Інтеграція з іншими виробниками та IoT девайсами	+	-	+	+	-
Використання алгоритмів комп'ютерного зору та машинного навчання для зчитування жестів	+	-	-	-	+

Аналізуючи представлену інформацію, можна зробити висновок, що наша система має мати суттєві переваги в аспектах кросплатформенності, зручності управління, відкритості архітектури та можливості гнучкого масштабування.

1.2 Аналіз переваг і недоліків існуючих рішень

Розглянуті в попередньому підрозділі системи мають як сильні сторони, так і певні обмеження, що важливо враховувати при створенні власного рішення для управління IoT-пристроями.

Система Swipe від Fibaro вирізняється високим рівнем інтеграції з іншими популярними брендами та платформами (Philips, D-Link, SmartThings, HomeKit тощо). Це дозволяє користувачу будувати гнучке середовище «розумного дому» з великою кількістю керованих пристроїв. Однак вона має низку недоліків: закритий API обмежує можливості глибокої кастомізації, а фізичне розміщення планшета лише на горизонтальній поверхні суттєво знижує зручність у використанні.

У системі WePe від Maxustech ключовою особливістю є використання сонара як основного елемента управління. Це рішення має потенціал для безконтактного керування, однак, як і в Swipe, для взаємодії користувач повинен знаходитися в безпосередній близькості до пристрою. Це знижує масштабованість у межах одного приміщення або квартири.

Технологія Leap Motion демонструє високу точність зчитування жестів завдяки інфрачервоному скануванню простору. Її перевага — детальне відстеження рухів у 3D-просторі. Водночас, пристрій має обмежену зону дії і потребує безперешкодного поля зору для коректної роботи. Це створює певні труднощі при спробах інтеграції в різноманітні конфігурації інтер'єру.

Кільце Nod Gesture Control забезпечує зручність і мобільність — користувач може здійснювати жести, не обмежуючись фіксованим місцем у просторі. Проте, необхідність постійного носіння спеціального аксесуару на руці може викликати дискомфорт у повсякденному використанні.

Загалом, основними недоліками описаних рішень є:

— обмежена зона дії або необхідність перебування поруч із пристроєм;

— обмежена можливість розширення системи або додавання нових пристроїв;

— закриті API, що унеможливлює або ускладнює розробку власного програмного забезпечення;

— потреба у використанні додаткових фізичних пристроїв, як-от кільця або сенсори, які не завжди зручні у використанні.

Отже, розробка власної системи повинна враховувати ці обмеження й прагнути усунути наявні недоліки, зробивши управління IoT-пристроями більш доступним, масштабованим і інтуїтивно зрозумілим.

1.3 Обґрунтування вибору архітектури системи

Під час проєктування комп'ютерної системи керування IoT-пристроями основна увага приділялася досягненню оптимального балансу між масштабованістю, простотою інтеграції та мінімізацією затримок у передачі команд. Саме тому було прийнято рішення реалізувати архітектуру на базі парадигми «клієнт–брокер–пристрій» із використанням протоколу MQTT, що спеціально створений для ресурсозалежних систем і відмінно підходить для Internet of Things.

MQTT (Message Queuing Telemetry Transport) було обрано як основний засіб обміну даними, оскільки він має низьке енергоспоживання, підтримує QoS-рівні доставки повідомлень, дозволяє ефективно організовувати підписку на події та забезпечує надійність навіть при нестабільному з'єднанні. У порівнянні з HTTP, MQTT має перевагу в умовах обмеженої пропускної здатності, що характерно для бездротових мереж, якими користуються більшість IoT-пристроїв.

Вибір платформи Android для клієнтського застосунку пояснюється високою популярністю цієї ОС серед користувачів смартфонів, а також широкими можливостями доступу до сенсорів пристрою, камери, Bluetooth-інтерфейсу та гнучким налаштуванням дозволів. Це дозволяє реалізувати інтуїтивний інтерфейс керування з мінімальною затримкою від зчитування жесту до виконання команди пристроєм.

Як обчислювальний модуль для керованого IoT-пристрою обрано мікроконтролер ESP32, який поєднує в собі велику обчислювальну потужність, підтримку Bluetooth Low Energy та Wi-Fi, а також має вбудовану файлову систему, що дозволяє зберігати конфігураційні параметри навіть після перезапуску. Це рішення ідеально підходить для розумних пристроїв, які повинні швидко реагувати на команди користувача та працювати в автономному режимі.

Крім того, архітектура передбачає можливість додавання кількох IoT-пристроїв, які підписуються на відповідні топіки MQTT-брокера. Завдяки цьому реалізується розширення системи без необхідності змінювати її загальну структуру. Це відкриває шлях до майбутнього масштабування: можливість додавати нові типи пристроїв — лампи, датчики, камери тощо — без значного перепроектування клієнтського додатку або змін у коді вже наявних компонентів.

Отже, обрана архітектура задовольняє всі технічні та функціональні вимоги до сучасної IoT-системи: простота інтеграції, кросплатформеність, енергоефективність, гнучкість у розгортанні та подальшому масштабуванні.

1.4 Принцип функціонування запропонованої системи

Програмна частина розробленої системи складається з двох основних компонентів: мобільного застосунку, який виконує роль клієнтського інтерфейсу, та IoT-пристрою, оснащеного мікроконтролером ESP32. Спілкування між компонентами здійснюється через мережу Інтернет за допомогою протоколу MQTT, а зчитування жестів реалізовано за допомогою бібліотеки комп'ютерного зору.

Для розпізнавання рухів руки використовується бібліотека MediaPipe Hands, розроблена Google. Вона дозволяє в режимі реального часу визначати положення до 21 ключової точки на руці користувача з високою точністю. Після отримання координат пальців проводиться аналіз просторових відстаней

між точками та формування одного з наперед визначених жестів (наприклад: "відкрита долоня", "кулак", "вказівний палець", тощо).

Щойно конкретний жест розпізнано, мобільний застосунок формує MQTT-повідомлення у форматі uuid#command, де:

— uuid — унікальний ідентифікатор пристрою, призначеного для обробки команди;

— command — код дії, що має бути виконана (наприклад, on, off, toggle).

MQTT-повідомлення публікується у відповідний топік, підписаним на який є всі активні IoT-пристрої. Однак лише той пристрій, що має відповідний uuid, виконує команду — це дозволяє уникнути конфліктів і забезпечити адресну доставку.

На початку роботи користувач підключається до IoT-пристрою за допомогою Bluetooth. Через зручний інтерфейс застосунку передаються параметри Wi-Fi мережі та MQTT-брокера. Після отримання даних пристрій самостійно підключається до мережі та реєструється на MQTT-сервері. Якщо процедура завершилася успішно, на застосунок надходить підтвердження, після чого Bluetooth-модуль вимикається для зниження енергоспоживання.

Процес ініціалізації зображено на рис. 1.5.

Після успішного налаштування, користувач переходить до головного меню, де відображаються всі доступні пристрої. При виборі певного пристрою запускається модуль розпізнавання жестів. Система очікує на зчитування одного з попередньо навчених жестів, після чого виконує дію, асоційовану з ним.

Особливістю реалізації є автоматичне розпізнавання руху руки, що дозволяє уникнути зайвих дотиків до екрана під час роботи. Це створює додатковий комфорт для користувача, адже всі дії виконуються безконтактно — лише за допомогою жестів, схожих на взаємодію з віртуальними об’єктами.

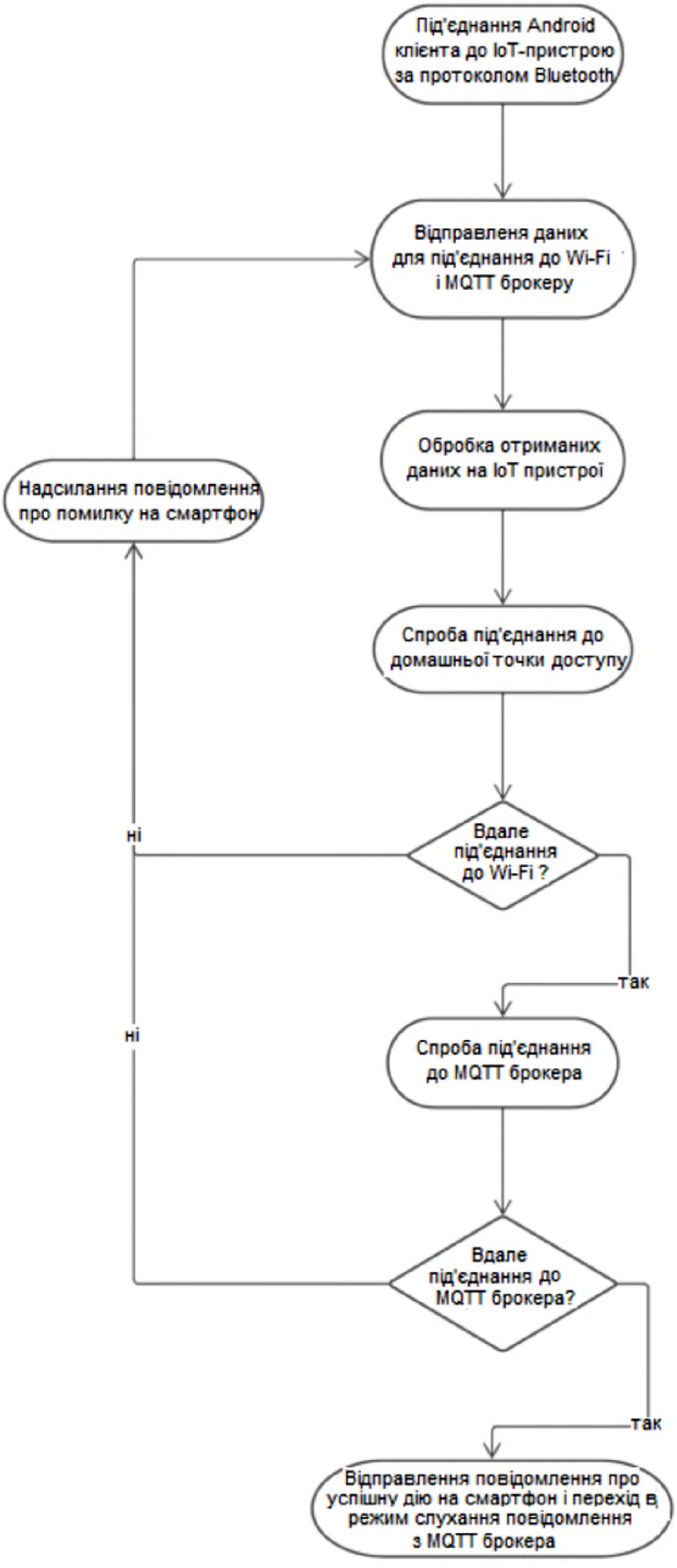

Рисунок 1.5 — Процес ініціалізації IoT пристрою

На UML-діаграмі нижче (рис. 1.6) наведено загальну послідовність взаємодії між мобільним клієнтом, MQTT брокером та IoT-пристроєм під час виконання команди.

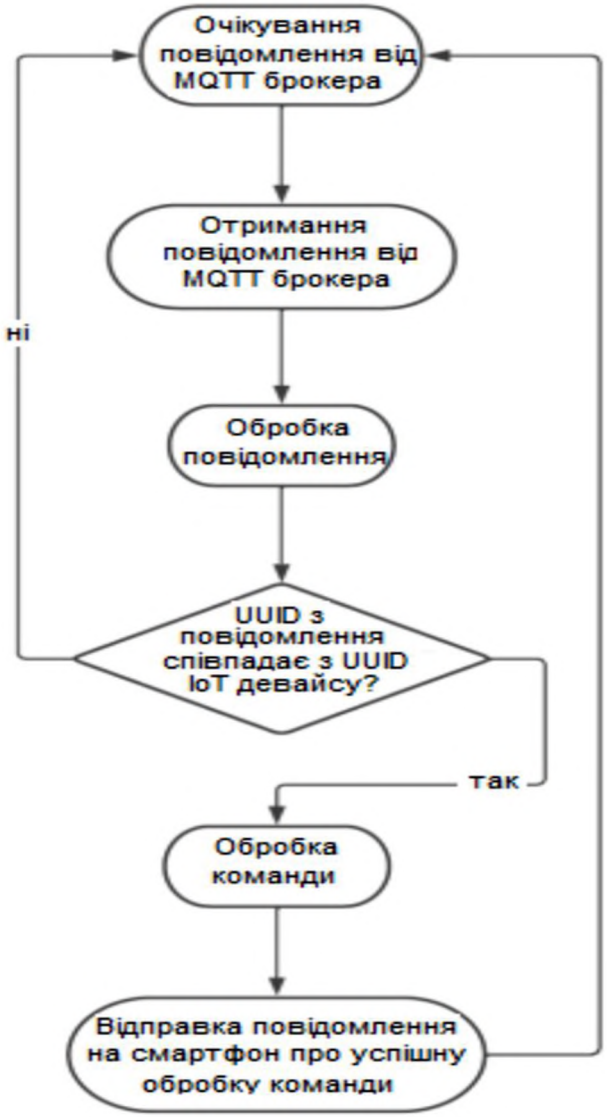

Рисунок 1.6 — Послідовність взаємодії між мобільним клієнтом, MQTT брокером та IoT-пристроєм

Завдяки використаним технологіям, система демонструє високу стабільність, швидку реакцію на дії користувача та здатність масштабуватися до десятків і сотень пристроїв без необхідності принципових змін у її архітектурі чи логіці взаємодії.

РОЗДІЛ 2 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ СИСТЕМИ

2.1 Технології виявлення жестів у потоці відеоданих

З метою вибору оптимального інструменту для зчитування жестів у потоковому відео було здійснено аналіз наявних технічних рішень. У процесі дослідження було виокремлено кілька основних підходів до реалізації розпізнавання жестів у відеоданих:

— застосування бібліотеки Chinese-number-gestures-recognition;

— створення індивідуальної моделі для розпізнавання жестів;

— використання бібліотеки HandWave;

— впровадження бібліотеки MediaPipe від компанії Google.

Бібліотека Chinese-number-gestures-recognition призначена виключно для виявлення жестів, що базуються на кількості піднятих пальців руки користувача.

Розробка власної моделі є складним процесом, що передбачає збір та анотування навчальних даних, а також проєктування відповідної архітектури моделі, здатної ефективно обробляти жести для конкретного завдання.

Бібліотека HandWave, попри свій функціонал, наразі вважається застарілою через відсутність підтримки сучасних версій операційної системи Android.

MediaPipe є найсучаснішим рішенням, яке дозволяє ефективно реалізувати методи машинного навчання в Android-додатках. Окрім Android, ця бібліотека підтримується також в iOS, десктопних застосунках на Python та вбудованих системах з використанням C++, що робить її особливо привабливою для використання у поточному проєкті.

MediaPipe надає комплекс інструментів і алгоритмів для обробки відео- та аудіопотоків у реальному часі. Зокрема, вона дозволяє ідентифікувати контури пальців завдяки поєднанню технологій комп’ютерного зору та глибинного навчання.

Розглянемо ключові етапи функціонування MediaPipe під час розпізнавання контурів пальців рук:

— захоплення відео: спочатку здійснюється отримання відеопотоку за допомогою камери або іншого пристрою. MediaPipe забезпечує підтримку різних форматів відео й джерел, зокрема вебкамер та відеофайлів;

— обробка кадрів: кожне відеозображення передається до системи обробки, яка застосовує алгоритми комп'ютерного зору та нейронні мережі для аналізу;

— виявлення ключових точок: за допомогою глибинних моделей виконується детекція характерних точок руки — зокрема, кінців пальців, суглобів тощо. Ці точки дозволяють точно ідентифікувати положення елементів кисті;

— побудова контурів: на основі знайдених ключових точок відбувається формування контурів пальців із використанням методів апроксимації або структурних зв'язків між точками;

— інтерпретація результатів: після побудови контурів результати аналізуються згідно з вимогами застосунку — наприклад, для розпізнавання певних жестів або визначення орієнтації пальців.

Для досягнення точного розпізнавання контурів пальців у MediaPipe застосовується кілька передових алгоритмів:

— MobileNet – легка та ефективна модель нейронної мережі, призначена для мобільних пристроїв, яка часто використовується для виявлення об'єктів та ключових точок на зображенні;

— SSD (Single Shot MultiBox Detector) – алгоритм, що дозволяє одночасно визначати класи об'єктів і координати обмежувальних рамок. У межах розпізнавання рук він виконує первинну детекцію і локалізацію;

— Landmark Estimation – методика оцінювання ключових точок, що поєднує згорткові нейронні мережі та регресійні моделі для передбачення координат важливих точок руки;

— Hand Tracking – технологія трекінгу, яка дозволяє відстежувати положення рук у режимі реального часу, забезпечуючи стабільність розпізнавання незалежно від зміни орієнтації або положення руки в кадрі.

2.2 Методи побудови зв'язку між клієнтським застосунком і IoT-пристроями

У процесі розробки системи керування пристроями Інтернету речей (IoT) особливу увагу слід приділити вибору протоколу обміну повідомленнями між клієнтським застосунком і підключеними пристроями. Від обраного способу комунікації залежить ефективність, надійність і масштабованість всієї системи. Існує низка протоколів, які спеціально створені або адаптовані для роботи з пристроями в розподілених мережах, зокрема у сфері розумного будинку. Нижче розглянемо основні з них.

AMQP (Advanced Message Queuing Protocol) — це промисловий стандарт протоколу обміну повідомленнями, орієнтований на забезпечення надійної, гарантованої доставки повідомлень. Він надає широкі можливості для організації черг, обробки підписок, фільтрації та маршрутизації повідомлень. AMQP переважно застосовується в корпоративних мережах або складних багатокомпонентних системах, де важлива структурована доставка даних. Протокол є досить потужним, але може виявитися надлишковим для простих IoT-систем із низьким енергоспоживанням.

CoAP (Constrained Application Protocol) — легковаговий протокол, призначений для пристроїв з обмеженими ресурсами. Працює поверх протоколу UDP, що забезпечує мінімальну затримку та споживання енергії. CoAP підтримує моделі запит-відповідь, спостереження за ресурсами та використовується в сенсорних мережах, де обсяг переданих даних має бути мінімальним. Завдяки своїй простоті, CoAP добре підходить для мікроконтролерів і пристроїв з автономним живленням.

XMPP (Extensible Messaging and Presence Protocol) — це протокол реального часу, спочатку розроблений для обміну миттєвими повідомленнями та керування присутністю користувачів у мережі. У контексті IoT він може бути адаптований для передачі команд, станів та сигналів між пристроями. XMPP забезпечує розширюваність, але вимагає більшої кількості ресурсів у порівнянні з іншими спеціалізованими протоколами IoT.

WebSocket — протокол, що дозволяє встановити стійке двостороннє з'єднання між клієнтом та сервером. Завдяки підтримці повнодуплексної передачі даних, WebSocket особливо корисний у тих застосунках, де потрібна миттєва реакція системи або зворотний зв'язок у реальному часі. У системах розумного будинку WebSocket може бути використаний, наприклад, для потокового моніторингу даних з датчиків або відеоспостереження.

MQTT (Message Queuing Telemetry Transport) — один із найпопулярніших протоколів у сфері IoT. Його особливістю є використання моделі «публікація-підписка» (publish/subscribe), що дозволяє спростити архітектуру обміну даними та зменшити навантаження на пристрої. MQTT забезпечує мінімальне споживання трафіку, підтримує декілька рівнів якості обслуговування (QoS), а також легко масштабується. Завдяки своїм перевагам MQTT став стандартом де-факто в багатьох IoT-проєктах.

У рамках даної кваліфікаційної роботи для побудови зв'язку між клієнтським застосунком та IoT-пристроями було обрано саме MQTT. Основними причинами такого вибору є наявний практичний досвід інтеграції з MQTT-сумісними платформами, ефективність передачі повідомлень, адаптованість до обмежених ресурсів та активна підтримка спільнотою розробників. Нижче подано порівняльний аналіз MQTT з іншими протоколами.

MQTT має значно простішу реалізацію, особливо для IoT-пристроїв із обмеженими ресурсами. AMQP, хоча й функціонально потужніший, потребує більше обчислювальних ресурсів і складніший у налаштуванні.

AMQP більше підходить для фінансових, банківських і корпоративних систем з потребою у транзакційності та складній маршрутизації. MQTT краще пристосований для розподілених сенсорних мереж та смарт-пристроїв.

MQTT дозволяє досягати вищої швидкодії в умовах високого навантаження завдяки оптимізації на малий об'єм трафіку.

MQTT використовує асинхронну модель pub/sub, тоді як CoAP базується на запитах та відповідях (request/response). Це впливає на гнучкість у побудові архітектури.

Протокол транспортного рівня. MQTT працює через TCP, що забезпечує надійність передачі, у той час як CoAP використовує UDP, що робить його ефективнішим, але менш надійним.

Придатність до мереж. CoAP більше підходить для локальних мереж із низькою затримкою, тоді як MQTT краще працює через Інтернет або мобільні мережі.

XMPP спочатку був створений для чатів, а MQTT — спеціально для IoT. Це визначає відмінності в структурі повідомлень та оптимізації протоколу.

MQTT створює менше навантаження на мережу та споживає менше енергії, що критично для мобільних або автономних пристроїв.

XMPP підтримує більшу кількість розширень (XEP), але це робить його важчим у реалізації на слабких пристроях.

WebSocket забезпечує двосторонній потік даних безпосередньо між клієнтом і сервером, що підходить для миттєвих подій. MQTT дозволяє передавати повідомлення через брокер, що забезпечує кращу масштабованість.

MQTT має вбудовані механізми QoS, збереження повідомлень та повторної доставки, тоді як при використанні WebSocket всі ці функції потребують окремої реалізації.

WebSocket переважно використовується в реальному часі (ігри, чати), тоді як MQTT — у розумних системах, де важлива стабільність та енергоефективність.

Таким чином, MQTT забезпечує найкращий баланс між простотою реалізації, гнучкістю, масштабованістю та низьким споживанням ресурсів, що робить його оптимальним вибором для розробки системи керування IoT-пристроями в рамках даного проєкту.

2.3 Структура та логіка побудови системи

Програмне рішення є комплексною системою, що складається з клієнтського застосунку та вбудованого IoT-пристрою, які взаємодіють між собою за допомогою обміну повідомленнями через MQTT-брокер. Такий підхід забезпечує просту, гнучку та масштабовану архітектуру, яка дозволяє ефективно інтегрувати нові пристрої та клієнтів без необхідності значної модифікації існуючої кодової бази.Загальна архітектура системи подана на рисунку 2.1.

У рамках цієї архітектури кожна квартира або об’єкт моніторингу асоціюється з окремим MQTT-топіком у брокері. Кожен топік функціонує як логічний канал зв’язку, до якого можуть підключатися як клієнтські пристрої, так і IoT-модулі. Обмін повідомленнями є двонаправленим: як клієнти, так і пристрої можуть надсилати та приймати повідомлення з відповідного топіка.

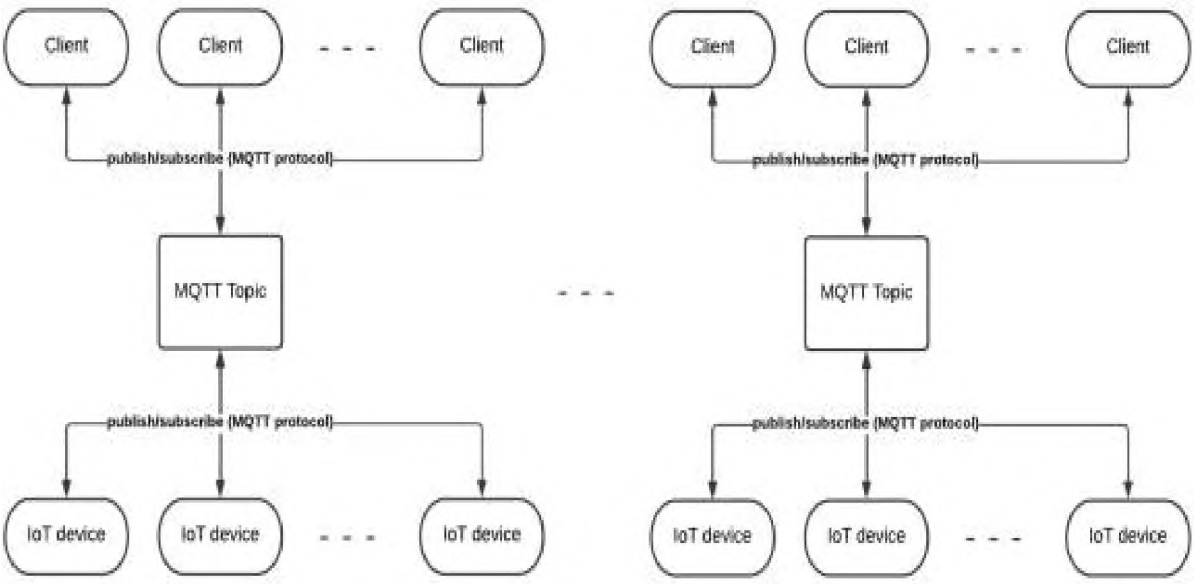

Рисунок 2.1 — Архітектура системи

У ролі клієнта може виступати будь-який пристрій, що має доступ до мережі, камеру та спеціалізоване програмне забезпечення. Формат переданих повідомлень — звичайний текст, що дозволяє уникати залежності від конкретних апаратних або програмних платформ. Завдяки цьому забезпечується високий рівень сумісності та гнучкості системи, а також можливість використання пристроїв з різними операційними системами та технічними характеристиками.

Використання MQTT як протоколу передачі даних дозволяє забезпечити:

— низький рівень споживання ресурсів мережі;

— високу швидкість обміну даними;

— можливість масштабування системи без зміни основної архітектури;

— стійкість до втрати з'єднання, завдяки механізмам повторної доставки повідомлень.

Архітектура є відкритою до розширення: до неї легко можуть бути додані нові типи IoT-пристроїв (сенсори руху, освітлення, датчики температури тощо) та нові клієнтські застосунки, включаючи мобільні, настільні або веб-інтерфейси, що суттєво підвищує універсальність рішення.

2.3.1 Архітектура мобільного застосунку на базі Android

Архітектура мобільного клієнта, розробленого для Android-платформи, побудована на основі підходу Clean Architecture (чистої архітектури), що дозволяє забезпечити модульність, масштабованість і тестованість програмного продукту. Основна мета цього підходу – створення чіткої структури, в якій кожен шар системи має свою відповідальність, а залежності між шарами строго регламентовані.

Загальна схема архітектури наведена на рисунку 2.2.

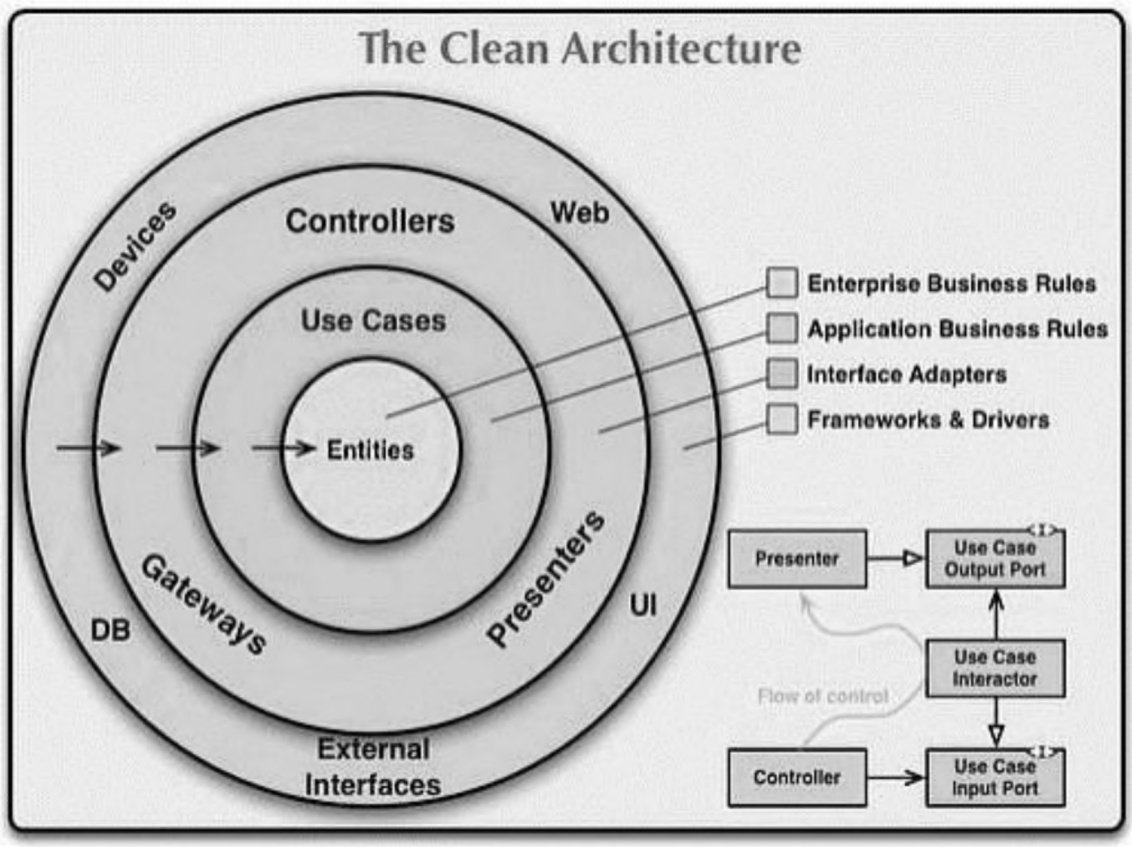

Рисунок 2.2 — Чиста архітектура Android-застосунку

Ключові принципи Clean Architecture:

— незалежність від фреймворків та бібліотек: бізнес-логіка відокремлюється від конкретних реалізацій, що спрощує заміну або оновлення технологічного стеку;

— розподіл на шари:

✓ Domain Layer — реалізує бізнес-логіку, основні сутності та інтерфейси, не залежить від зовнішніх API;

✓ Data Layer — відповідає за збереження та отримання даних із зовнішніх джерел (база даних, API, тощо);

✓ Presentation Layer — забезпечує інтерфейс користувача та взаємодію з domain-шаром через ViewModel;

✓ DI Layer — реалізує ін’єкцію залежностей, що полегшує керування залежностями в системі;

— принцип єдиної відповідальності: кожен компонент виконує лише одну функцію, що спрощує внесення змін;

— чіткі межі між шарами: шари взаємодіють виключно через інтерфейси, що підвищує модульність системи.

На рівні presentation layer реалізація побудована із використанням архітектурного патерну MVI (Model–View–Intent), загальна схема якого показана на рисунку 2.3.

Патерн MVI передбачає реалізацію односпрямованого потоку даних та має наступні складові:

— Model — шар даних, що містить необхідну інформацію для роботи застосунку;

— View — інтерфейс користувача, що реагує на зміну стану та передає події (Intents) у бізнес-логіку;

— Intent — користувацькі дії, що описують бажану зміну стану (натискання кнопки, введення тексту тощо);

— State — незмінний об’єкт, що представляє поточний стан інтерфейсу;

— Interactor — обробляє наміри, виконує бізнес-операції та повертає оновлений стан.

Переваги MVI-архітектури:

— забезпечує передбачувану логіку завдяки однонаправленому потоку даних;

— підвищує стабільність застосунку завдяки використанню незмінних об’єктів;

— покращує тестованість, оскільки всі компоненти розділені та можуть перевірятися ізольовано.

Недоліки MVI-архітектури:

Складність реалізації: MVI вимагає більшого початкового налаштування, ніж інші архітектури (як-от MVC чи MVVM), особливо на невеликих проєктах. Розробникам треба продумати інтенти, редьюсери, стейт-менеджмент.

Велика кількість коду: Через явну передачу станів і дій (Intent → Model → ViewState) виникає багато шаблонного коду, який може ускладнити підтримку проєкту.

Затримки у відображенні: Оскільки UI оновлюється лише у відповідь на новий ViewState, можуть бути затримки між дією користувача і оновленням інтерфейсу, якщо обробка складна або не оптимізована.

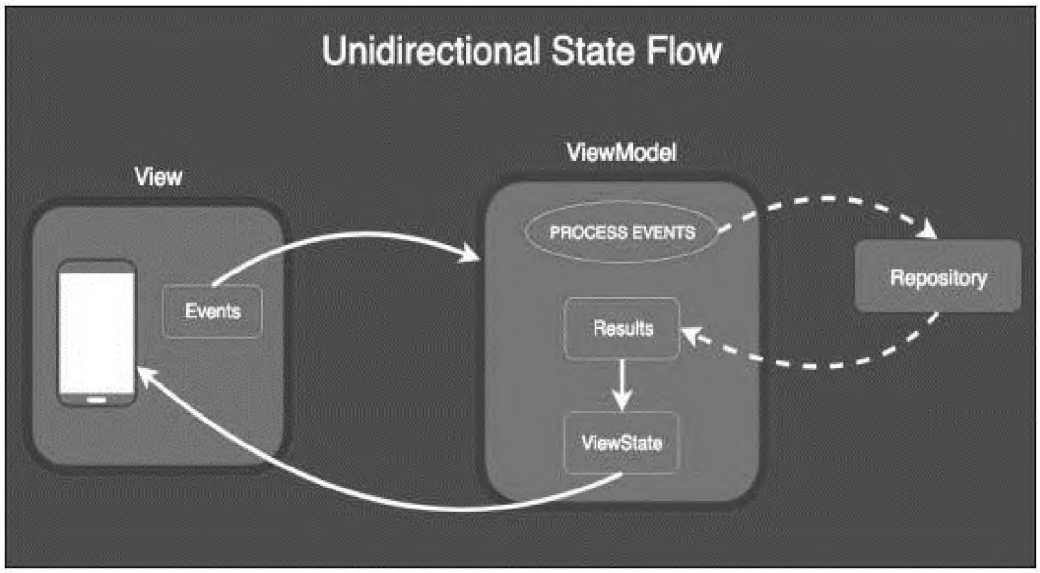

Рисунок 2.3 — Архітектура патерну MVI

У реалізації застосунку:

— View виступають Activity або Fragment;

— Interactor реалізовано через ViewModel;

— Model реалізується в domain та data шарах відповідно до принципів Clean Architecture.

У порівнянні з іншими підходами:

— на відміну від MVP (Model-View-Presenter), MVI має чіткіший контроль за станом інтерфейсу;

— у порівнянні з MVVM (Model-View-ViewModel), MVI краще ізолює потоки даних та зменшує ризик непередбачуваної поведінки.

Таким чином, обрана архітектура клієнта поєднує в собі переваги сучасних підходів до розробки Android-застосунків, забезпечує масштабованість, надійність та зручність підтримки системи.

2.3.2 Структура апаратного модуля IoT-пристрою

Як уже зазначалося раніше, центральним апаратним елементом IoT-пристрою є мікроконтролер ESP32, який забезпечує основну обчислювальну логіку, а також засоби комунікації через Bluetooth і Wi-Fi. Програмування цього мікроконтролера відбувається шляхом створення єдиного файлу з кодом, відомого як скетч (англ. sketch) [9]. У цьому файлі реалізується весь алгоритм роботи пристрою, що включає взаємодію з користувачем, підключення до мережі та обробку MQTT-повідомлень.

Для реалізації функціоналу IoT-пристрою був створений скетч, фрагмент якого подано в додатку Б. Структура програмного коду організована таким чином, що робота пристрою умовно поділяється на два основні режими функціонування:

— Bluetooth-режим;

— Wi-Fi-режим.

У Bluetooth-режимі ESP32 очікує вхідного з'єднання від мобільного пристрою (клієнта). Після встановлення Bluetooth-з'єднання, мікроконтролер переходить у режим очікування налаштувань – зокрема, параметрів доступу до Wi-Fi-мережі та MQTT-брокера. Таким чином, початкове налаштування пристрою здійснюється бездротово, без потреби фізичного підключення до комп'ютера або зміни прошивки.

Після отримання всіх необхідних параметрів ESP32 ініціює підключення до вказаної Wi-Fi-мережі. Якщо з'єднання успішне, пристрій автоматично переходить у Wi-Fi-режим, де здійснює підключення до MQTT-брокера. У цьому режимі мікроконтролер постійно очікує на вхідні повідомлення з відповідного MQTT-топіка та реагує на них згідно з логікою, закладеною в

програмному коді. Така логіка може включати керування зовнішніми пристроями, збирання даних з сенсорів, активацію сигналізації тощо.

На рисунку 2.4 схематично зображено архітектуру та процес роботи IoT-девайсу.

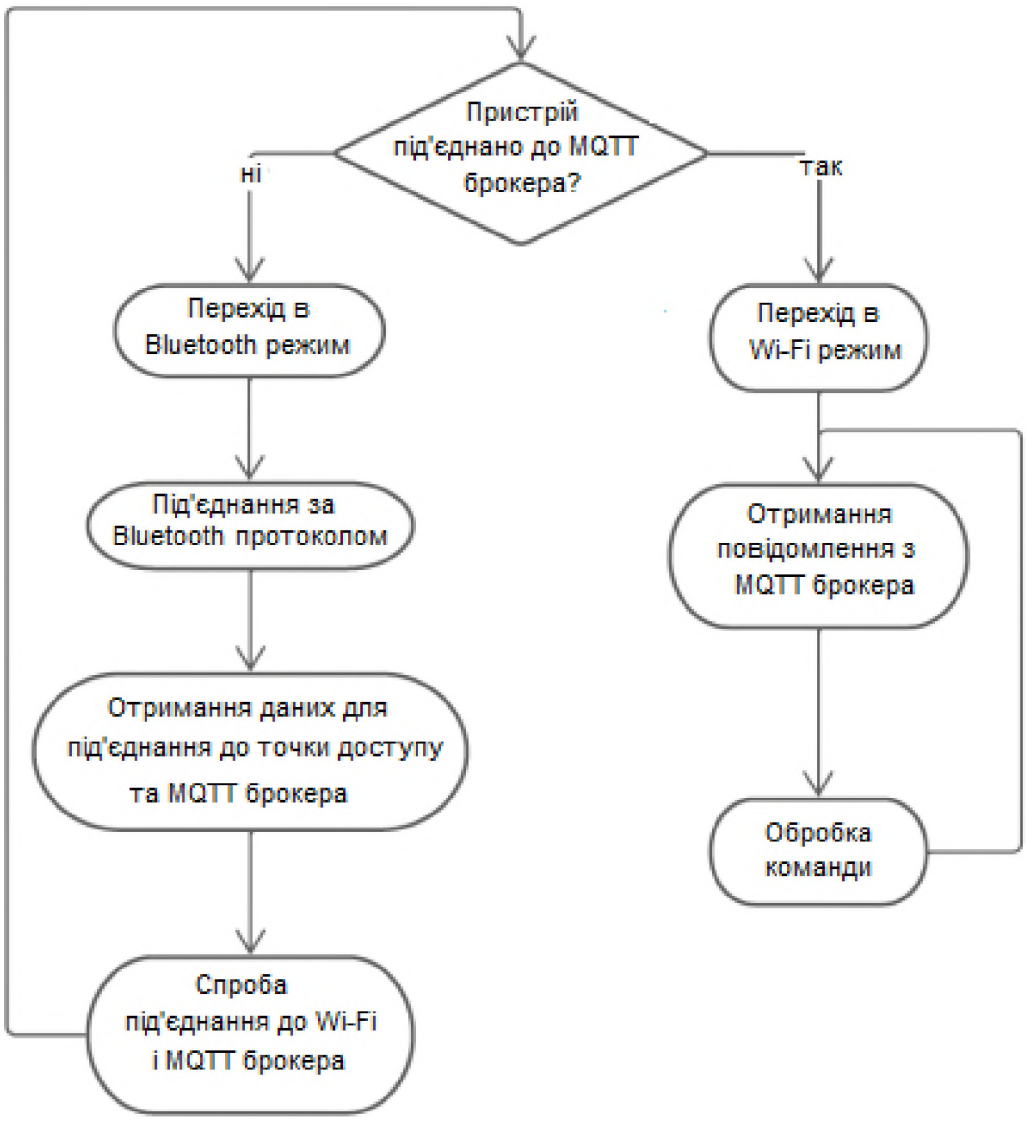

Рисунок 2.4 — Принцип роботи IoT пристроя

Обрана архітектура дозволяє досягти таких переваг:

— мобільність налаштування — завдяки Bluetooth-підключенню користувач може змінювати параметри пристрою безпосередньо зі смартфона;

— безперервна робота в мережі — після первинного налаштування пристрій автоматично переходить до стабільного режиму роботи через Wi-Fi;

— гнучкість логіки взаємодії — завдяки MQTT-протоколу пристрій може як приймати, так і надсилати повідомлення, що відкриває широкі можливості інтеграції у більші IoT-системи.

Таким чином, архітектура IoT-пристрою забезпечує гнучке, зручне у використанні та масштабоване рішення, яке повністю відповідає вимогам системи з дистанційною взаємодією через Інтернет речей.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Впровадження Android-застосунку

3.1.1 Проєктування інтерфейсу користувача

Реалізація клієнтського Android-застосунку розпочалася з розробки інтерфейсу користувача, який є ключовим елементом взаємодії між користувачем і програмною системою. Інтерфейс було створено з використанням стандартних інструментів Android SDK, зокрема компонентів Activity та Fragment, що є базовими одиницями побудови візуального інтерфейсу у середовищі Android [10], [11].

Для розмітки інтерфейсів застосовано XML-мову розмітки, що дозволяє чітко визначити структуру, стилі та поведінку елементів UI. Такий підхід дозволяє відокремити логіку від візуального оформлення, що спрощує подальшу підтримку та масштабування застосунку.

Окрім традиційного підходу на основі XML, розглядалася також можливість використання сучасного підходу до побудови інтерфейсів — Jetpack Compose. Це декларативна технологія створення UI від Google, яка дозволяє описувати компоненти інтерфейсу безпосередньо в коді на Kotlin, використовуючи компактні та зручні конструкції [12], [13]. Такий підхід суттєво зменшує кількість коду, підвищує читабельність та прискорює розробку.

Проте, незважаючи на переваги Jetpack Compose, зокрема його інтегровану реактивність і зручність при роботі з сучасними Android-архітектурами, ця технологія поки що має обмеження у взаємодії з окремими компонентами пристрою, зокрема з камерою смартфона. Це обмеження зумовлено тим, що багато існуючих бібліотек та API ще не повною мірою адаптовані до Compose-інтерфейсів [14].

У зв’язку з необхідністю реалізації функціоналу, пов’язаного з розпізнаванням жестів за допомогою камери (див. п. 3.1.2), було прийнято рішення залишитися на перевіреному рішенні з використанням XML-розмітки.

Такий підхід забезпечує стабільну роботу з Camera API, а також дозволяє реалізувати гнучке компонування елементів керування, таких як кнопки, поля введення, індикатори статусу тощо.

Розроблений інтерфейс охоплює такі основні екрани:

— головний екран — відображає поточний статус підключення до IoT-девайсу та MQTT-брокера;

— екран налаштувань — призначений для первинного введення параметрів Wi-Fi та MQTT;

— екран управління пристроєм — надає змогу взаємодіяти з IoT-пристроєм за допомогою жестів або кнопок;

— сервісний екран — містить інформацію про роботу системи, лог з'єднань тощо.

Таким чином, вибір XML-технології та компонентів Activity/Fragment обумовлений технічними вимогами до системи та забезпечує надійність, стабільність і функціональну повноту реалізованого інтерфейсу користувача.

3.1.2 Технологія розпізнавання жестів з камери смартфона

Функціональність розпізнавання жестів реалізована за допомогою бібліотеки MediaPipe, розробленої компанією Google. Ця бібліотека надає готові модулі для обробки відеопотоку та виявлення різних ознак, зокрема ключових точок руки, які використовуються для побудови контурів долоні в режимі реального часу.

У застосунку розпізнавання жестів інтегровано через спеціально реалізований компонент – кастомну View, що відповідає за отримання відеопотоку з фронтальної камери смартфона та його передавання до MediaPipe для подальшої обробки. Після обробки відео, бібліотека повертає координати 21 ключової точки руки, які відповідають суглобам і кінчикам пальців. На основі цих точок створюється сітка-контур долоні, яка відображається поверх відеозображення. Приклад такої обробки представлено на рисунку 3.1.

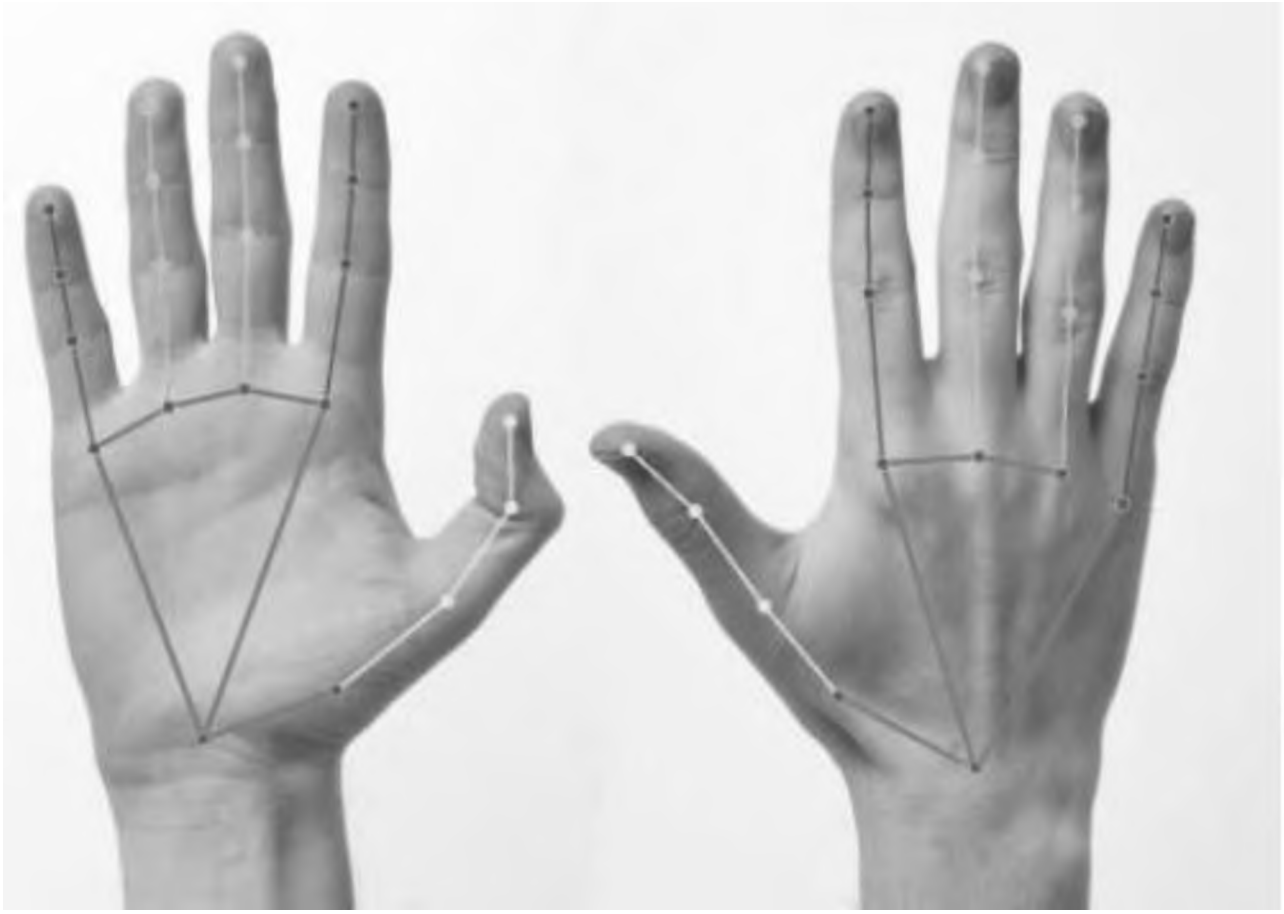

Рисунок 3.1 — Суть роботи бібліотеки

Отримані координати дозволяють виконувати геометричні розрахунки у 2D-просторі, зокрема визначати відстані між певними точками пальців. Це відкриває можливості для реалізації власних алгоритмів розпізнавання жестів шляхом аналізу просторового розташування пальців.

Для побудови логіки розпізнавання жестів у реалізованому застосунку використано власний емпіричний підхід: на основі дослідного калібрування було встановлено значення відстаней, які відповідають певним жестам. Розрахунок виконується за допомогою стандартної евклідової формули для відстані між двома точками на площині:

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

де x1, y1 та x2, y2 — координати двох вибраних ключових точок.

У випадку реалізованої системи, ключовими є вказівний та середній пальці. Якщо відстань між їхніми кінчиками менша за заздалегідь встановлене порогове значення, система інтерпретує це як команду вимкнення пристрою. У протилежному випадку — коли пальці розведені — формується команда на вмикання пристрою. Такий підхід дозволяє реалізувати інтуїтивне керування IoT-девайсом за допомогою жестів без необхідності дотиків до екрана.

Таким чином, застосування MediaPipe у поєднанні з власною обробкою координат точок забезпечує ефективну та гнучку систему розпізнавання жестів, що є ключовим елементом реалізованого інтерфейсу управління.

3.1.3 Першочергове налаштування IoT-пристрою через клієнтський додаток

Початкове налаштування пристрою відбувається шляхом обміну даними між клієнтським застосунком та IoT-девайсом за допомогою протоколу Bluetooth. Це дозволяє забезпечити локальне з'єднання між смартфоном користувача та мікроконтролером ще до моменту надання останньому доступу до інтернету. З боку клієнта для реалізації взаємодії з модулем Bluetooth смартфона було створено клас BluetoothController, який через інтерфейс Android Framework забезпечує повноцінну роботу з Bluetooth-модулем пристрою. Цей клас надає функціональні можливості для сканування доступних поблизу пристроїв, встановлення з’єднання з обраним девайсом, а також для надсилання та отримання повідомлень у рамках встановленого Bluetooth-з’єднання.

Діаграма класів, пов’язаних із роботою BluetoothController, подана на рис. 3.2.

Перед детальним розглядом архітектури Bluetooth-зв’язку доцільно пояснити низку понять, які стосуються як Android Framework, так і сторонніх бібліотек, що використовуються у застосунку.

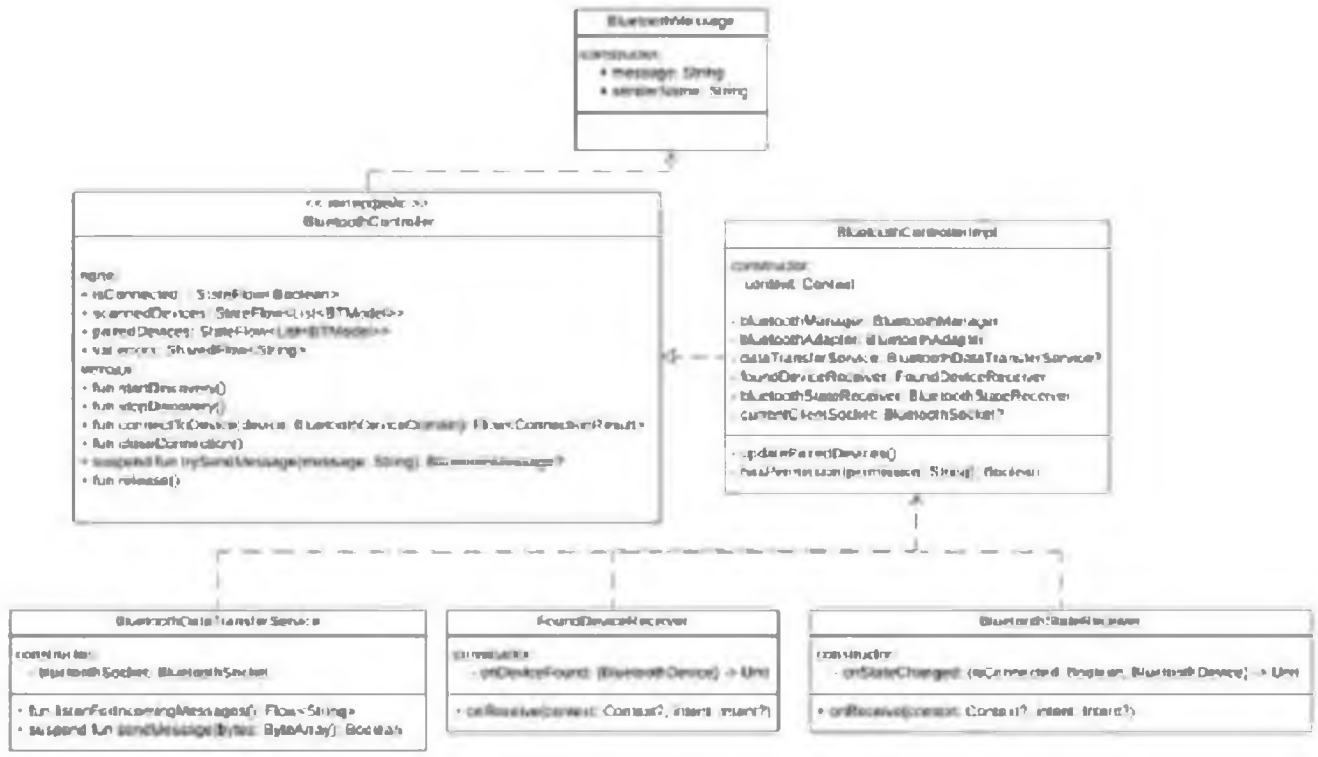

Рисунок 3.2 — Діаграма класів Bluetooth

Flow — клас, що постачається в складі бібліотеки Kotlin Coroutines. Його призначенням є реалізація реактивної моделі передачі даних. Завдяки Flow можна організувати асинхронну та ефективну взаємодію між джерелом даних і підписниками, що особливо актуально в контексті отримання повідомлень від пристрою без блокування головного потоку виконання.

BroadcastReceiver — компонент Android Framework, що використовується для перехоплення системних або прикладних широкомовних повідомлень (broadcast). Це дозволяє реагувати на зовнішні або внутрішні події, зокрема початок або завершення сканування пристроїв, зміну стану модуля Bluetooth, втрату або встановлення з’єднання тощо.

BluetoothSocket — клас, який інкапсулює механізм обміну даними між двома пристроями через Bluetooth. Він забезпечує доступ до потоків inputStream та outputStream, які використовуються для прийому та передачі повідомлень відповідно. Саме через цей сокет реалізується безпосереднє з'єднання клієнта та сервера.

BluetoothAdapter — основний клас, який надає доступ до функціональності Bluetooth модуля на рівні системи. Його методи дозволяють увімкнути або вимкнути Bluetooth, розпочати пошук пристроїв, отримати список вже спарених девайсів, а також ініціювати з'єднання за допомогою UUID-сервісів.

BluetoothDataTransferService — окремий сервіс, реалізований для забезпечення передачі даних після успішного встановлення з’єднання. Метод listenForIncomingMessages() створює реактивний потік Flow, який постійно зчитує байтові дані з вхідного потоку inputStream, перетворює їх у текстовий формат UTF-8 та передає далі у вигляді подій для інтерфейсу користувача. Метод sendMessage() виконує зворотну операцію: конвертує введений користувачем текст у байти та надсилає їх до IoT-пристрою через outputStream.

FoundDeviceReceiver — підклас BroadcastReceiver, що активується під час сканування та відповідає за обробку інформації про знайдені пристрої. Він фільтрує отримані дані та передає перелік доступних Bluetooth-девайсів у BluetoothControllerImpl.

BluetoothStateReceiver — ще один нащадок BroadcastReceiver, що відслідковує зміну стану Bluetooth-з'єднання: наприклад, коли з’єднання з пристроєм було встановлено або втрачено. Ця інформація також передається до контролера та відображається в інтерфейсі користувача.

BluetoothMessage — клас-обгортка для повідомлень, які передаються в системі. Він дозволяє класифікувати повідомлення за типом (вхідне/вихідне), зберігати вміст та метадані, необхідні для їх подальшої обробки або виведення.

Інтерфейс BluetoothController визначає контракт між логікою застосунку (ViewModel) та його Bluetooth-реалізацією. Основні його поля й методи:

— isConnected: Flow<Boolean> — реактивний індикатор стану з’єднання;

— scannedDevices: Flow<List<BluetoothDevice>> — потік, що містить список знайдених пристроїв;

— pairedDevices: List<BluetoothDevice> — список пристроїв, які вже були раніше під’єднані;

— errors: Flow<String> — потік повідомлень про помилки.

Основні методи:

— startDiscovery() — розпочинає пошук доступних Bluetooth-пристроїв;

— stopDiscovery() — завершує поточне сканування;

— connectToDevice(device: BluetoothDevice) — встановлює з’єднання з вибраним пристроєм;

— sendMessage(message: String) — надсилає текстове повідомлення через Bluetooth;

— closeConnection() — закриває активне з’єднання;

— release() — скидає стан контролера до початкового.

Загальна логіка сценарію взаємодії реалізована таким чином, щоб забезпечити інтуїтивно зрозумілий користувацький досвід. Після запуску мобільного застосунку користувач переходить на екран додавання нового пристрою, де автоматично запускається сканування. У процесі пошуку всі знайдені Bluetooth-девайси відображаються у списку. Після вибору потрібного пристрою система ініціює встановлення з’єднання.

Якщо з’єднання встановлено успішно, користувач переходить на наступний екран, де вводить параметри для налаштування IoT-пристрою: SSID та пароль Wi-Fi мережі, а також адресу та порт MQTT-брокера. Після підтвердження введених даних відбувається передача цієї інформації на IoT-девайс. Останній намагається підключитись до мережі та брокера.

У разі успіху користувач отримує повідомлення про успішне підключення пристрою до інтернету. В іншому випадку відображається повідомлення про помилку з пропозицією перевірити правильність введених параметрів або повторити спробу пізніше. Такий підхід забезпечує адаптивну та стійку до помилок систему початкової ініціалізації IoT-пристрою, придатну для користувачів без технічної підготовки.

3.1.4 Організація керування IoT-девайсом

Після завершення початкового налаштування IoT-пристрій автоматично з'являється у загальному переліку доступних девайсів на головному екрані мобільного застосунку. Цей екран слугує центральним вузлом управління всіма підключеними пристроями, дозволяючи користувачеві швидко орієнтуватися серед доступних об'єктів та вибирати той, з яким необхідно взаємодіяти. Інтерфейс цього екрану реалізовано з урахуванням принципів зручності та інтуїтивної зрозумілості: кожен пристрій представлено у вигляді окремої плитки з ідентифікатором, статусом підключення та відповідною іконкою.

Після вибору конкретного пристрою відкривається окремий екран з вбудованим відеопотоком з камери смартфона, який слугує інтерфейсом взаємодії з пристроєм за допомогою жестів руки. У цьому вікні реалізовано механізм візуального відстеження положення кисті користувача в реальному часі, що забезпечується алгоритмом комп'ютерного зору. Додатково на екрані розміщено довідковий блок у вигляді інтерактивної пам'ятки, яка містить список доступних жестів та відповідні дії, які вони ініціюють. Це дозволяє користувачу без попередньої підготовки або технічного досвіду швидко опанувати систему керування.

Виявлення жестів здійснюється відповідно до алгоритмічної логіки, описаної у попередньому підрозділі, з урахуванням просторових координат ключових точок на кисті руки. Після того, як певний жест успішно розпізнано, мобільний застосунок автоматично формує керуюче повідомлення у вигляді текстового пакету та ініціює його передачу до цільового IoT-пристрою через MQTT-протокол. Передача здійснюється асинхронно для забезпечення високої швидкості реагування та мінімізації затримок у мережі.

Для взаємодії з MQTT-брокером використовується сучасна клієнтська бібліотека HiveMQ MQTT Client [15], яка дозволяє реалізувати повноцінну підтримку протоколу MQTT на платформі Android. Щоб спростити використання цієї бібліотеки у проєкті та уникнути дублювання коду,

розроблено спеціалізований обгортковий клас MQTTClient, який інкапсулює базові операції та логіку роботи з мережею.

Діаграма структури цього класу представлена на рис. 3.3.

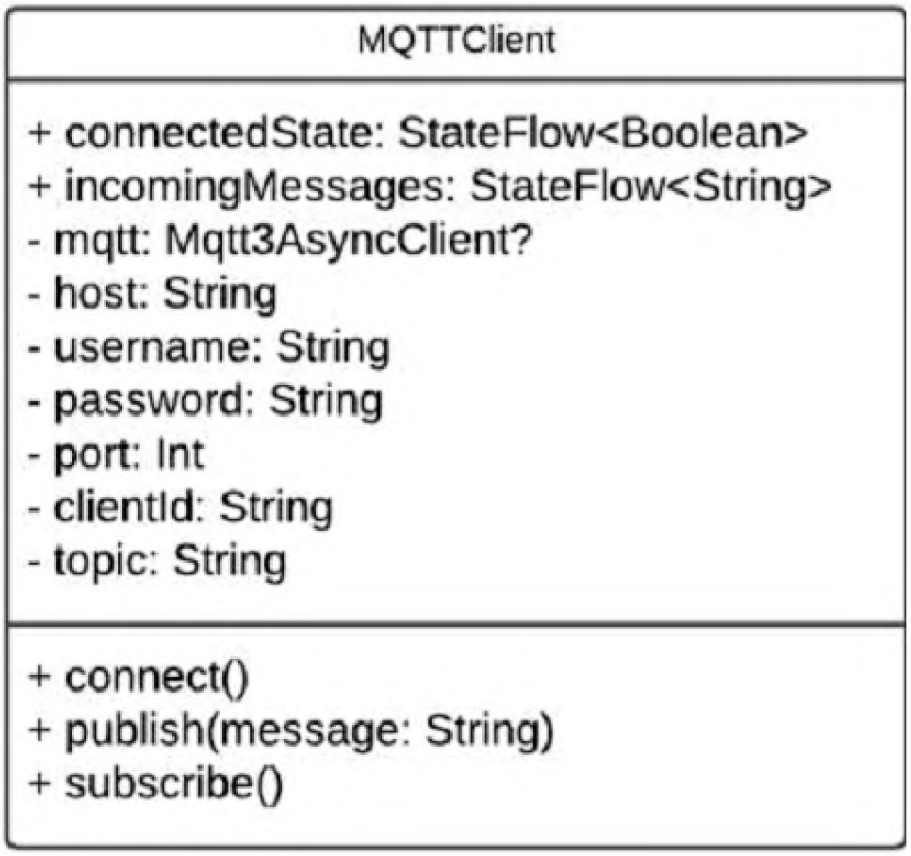

Рисунок 3.3 — Забраження діаграма класів MQTTClient

Клас MQTTClient відіграє роль програмного посередника, що забезпечує стабільний канал зв'язку між застосунком і сервером MQTT. Його архітектура орієнтована на реактивний підхід до обробки подій, що досягається завдяки використанню Kotlin Flow.

Основні поля класу включають:

— connectedState — об'єкт типу StateFlow<Boolean>, який повідомляє про поточний стан з'єднання з брокером. Це дозволяє вчасно реагувати на обриви зв'язку або помилки підключення;

— incomingMessages — потік об'єктів типу MQTTMessage, що представляють дані, отримані від брокера, і надходять із відповідних підписаних топіків;

— MQTT — клієнтський об’єкт з бібліотеки HiveMQ, який містить низькорівневу реалізацію усіх функцій MQTT-протоколу, таких як створення з’єднання, підписка, публікація тощо;

— службові поля serverUri, clientId, username, password, topics — містять конфігураційні параметри підключення до брокера та список топіків для роботи.

Клас має такі ключові методи:

— connect() — ініціює встановлення з’єднання з брокером MQTT, враховуючи параметри автентифікації, з’єднання по TLS, тайм-аути та інші налаштування. Після успішного з'єднання оновлюється connectedState;

— publish(topic: String, message: String) — дозволяє надсилати повідомлення до визначеного топіка. Наприклад, жест "увімкнути світло" транслюється у вигляді публікації до топіка device/lamp/power;

— subscribe(topic: String) — забезпечує підписку на отримання повідомлень із заданого топіка. Нові повідомлення автоматично надходять у incomingMessages, де можуть бути оброблені, наприклад, для оновлення інтерфейсу користувача або зворотного зв’язку з пристроєм.

Такий підхід дозволяє створити універсальний і повторно використовуваний механізм взаємодії з розподіленими IoT-пристроями, що спрощує масштабування системи на більшу кількість пристроїв без суттєвої модифікації клієнтського коду. Крім того, застосування бібліотеки HiveMQ забезпечує високу надійність, продуктивність та підтримку останніх стандартів MQTT, що є критично важливим для проєктів, орієнтованих на реальний час і розподілену архітектуру.

Окрім базової взаємодії через публікацію повідомлень, у системі реалізовано механізм зворотного зв’язку, що дозволяє користувачеві

отримувати підтвердження про виконання команди або поточний стан IoT-пристрою. Після отримання керуючого сигналу, пристрій може сформувати відповідь (наприклад, статус «виконано», «помилка», «виконання триває») та надіслати її у відповідний MQTT-топік, на який мобільний застосунок підписаний. Таким чином, забезпечується двостороння асинхронна комунікація, що дозволяє реалізувати принципи зворотного контролю, а також інформувати користувача про реальний стан пристрою в інтерфейсі програми.

Для гарантування захищеного обміну даними між мобільним застосунком та MQTT-брокером у MQTTClient реалізовано підтримку шифрування з'єднання за допомогою протоколу TLS (Transport Layer Security). Це дозволяє уникнути перехоплення або підміни керуючих команд у разі використання відкритих мереж. Крім того, підтримується автентифікація через логін і пароль, що захищає MQTT-брокер від несанкціонованого доступу.

Щоб забезпечити надійність та безперервність роботи системи, реалізовано механізми повторного підключення у разі втрати зв'язку, а також автоматичне оновлення інтерфейсу користувача відповідно до поточного стану з'єднання. У випадку критичних помилок з'єднання, програма повідомляє користувача через відповідне повідомлення у графічному інтерфейсі з рекомендаціями щодо усунення проблеми (перевірка мережі, перезапуск пристрою тощо).

Таким чином, розроблена система забезпечує не лише зручне й інтуїтивно зрозуміле керування IoT-пристроями за допомогою жестів, але й гарантує надійний, захищений і двосторонній канал зв'язку, що відповідає сучасним вимогам до розумних систем управління.

3.2 Реалізація апаратної частини IoT-рішення

На ринку присутня велика кількість готових IoT-пристроїв, однак у більшості з них внутрішній програмний інтерфейс (API) є закритим або обмеженим у доступі. Це значно ускладнює або унеможливлює інтеграцію

таких пристроїв у власну екосистему або керування з нестандартних програмних рішень. Саме тому, після обговорення із науковим керівником, було прийнято рішення створити власний IoT-девайс, що дозволяє повністю контролювати як його апаратну частину, так і програмну логіку.

У межах реалізації проєкту було обрано концепцію «розумної розетки» — пристрою, який підключається до електромережі та дає змогу дистанційно вмикати або вимикати подачу електроенергії на підключене до нього електрообладнання. Такий пристрій є поширеним прикладом побутового IoT-рішення, яке має практичну цінність і водночас є зручним для реалізації з точки зору апаратного й програмного забезпечення.

Основні вимоги до реалізованого пристрою:

— можливість керування через бездротовий канал зв'язку (Bluetooth або Wi-Fi);

— підтримка обміну повідомленнями через протокол MQTT;

— мінімальна затримка при виконанні команд користувача;

— можливість оновлення прошивки та розширення функціоналу;

— компактні габарити та безпечна конструкція для підключення до побутової мережі 220 В.

Для реалізації було використано мікроконтролер ESP32 як основний керуючий елемент. Цей чип поєднує в собі високу обчислювальну потужність, низьке енергоспоживання та вбудовані модулі Bluetooth і Wi-Fi, що робить його ідеальним для створення пристроїв з можливістю бездротового керування. До ESP32 було підключено реле, яке фізично замикає або розмикає ланцюг подачі напруги на вихідну розетку.

На мікроконтролер завантажено прошивку, написану мовою C++ із використанням середовища Arduino IDE. Вона реалізує прийом MQTT-повідомлень через мережу Wi-Fi, обробку команд на вмикання або вимикання реле, а також передачу зворотного сигналу про стан пристрою.

Для безпеки користування було додатково впроваджено:

— захист від повторного включення за короткий проміжок часу (дебаунс);

— контроль напруги на вході (використовуючи аналоговий вхід ESP32 та подільник напруги);

— ізоляцію силової частини схеми через оптопару для захисту мікроконтролера.

Таким чином, реалізована розумна розетка повністю відповідає вимогам до IoT-девайсу: вона підтримує інтеграцію в мобільний застосунок, має низький час відгуку, підтримує зворотній зв'язок та є безпечною для користування в домашніх умовах.

3.2.1 Застосування смарт-розетки в системі

Реалізований пристрій виконує функції керованого комутатора в електричному колі. У рамках виконання цієї роботи «розумна» розетка була інтегрована в один із фазових проводів стандартного побутового подовжувача. Завдяки цьому було забезпечено можливість апаратного керування подачею електроенергії на кінцеві пристрої, підключені до подовжувача.

Після вбудовування модуля в ланцюг, з використанням мікроконтролера з'являється змога віддалено змінювати стан електричного кола — замикати або розмикати його. Це відбувається шляхом керування електромагнітним реле, підключеним до мікроконтролера. Коли користувач взаємодіє з мобільним застосунком, генерується відповідна команда, яка передається через MQTT протокол на пристрій. Мікроконтролер, отримавши повідомлення з MQTT топіка, активує або деактивує реле, яке фізично замикає або розриває електричне коло.

Таким чином, реалізується повноцінне дистанційне керування подачею електроенергії на пристрої, що підключені до розетки. Це може бути корисним для автоматизації роботи побутової техніки, реалізації сценаріїв енергозбереження або з міркувань безпеки.

3.2.2 Електрична схема смарт-розетки

На рисунку 3.4 зображено загальну структурну електричну схему реалізованого пристрою.

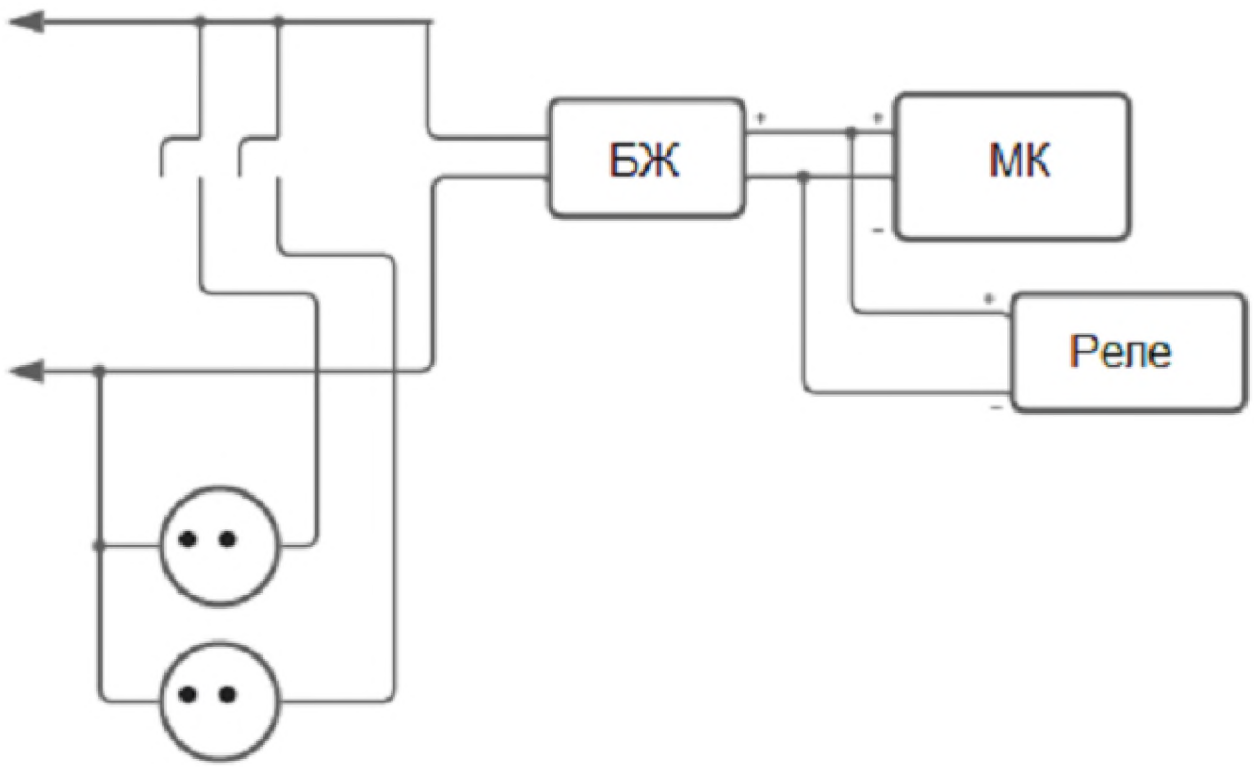

Рисунок 3.4 — Структурна схема системи

Як мікроконтролер було обрано модуль ESP32 [16], що є потужним та енергоефективним рішенням з вбудованими модулями Wi-Fi та Bluetooth. Наявність Wi-Fi інтерфейсу дозволяє пристрою підключатися до локальної мережі та обмінюватися даними з MQTT брокером, тоді як Bluetooth застосовується для початкового налаштування при підключенні до мобільного застосунку.

З огляду на те, що ESP32 функціонує при напрузі 3.3–5 В постійного струму, постала необхідність у зниженні напруги зі стандартних 220 В змінного струму до відповідного рівня. Для цього використовується модуль імпульсного джерела живлення Hi-Link [17], що забезпечує стабільну подачу 5 В постійного струму, придатну для живлення логіки пристрою.

Ключовим елементом у схемі є електромагнітне реле, що підключене до одного з цифрових виходів ESP32. Саме це реле виконує замикання або розмикання електричного кола, відповідно до вхідної команди, отриманої з MQTT топіка. Таким чином, мікроконтролер виступає посередником між користувацьким інтерфейсом (мобільним застосунком) та виконавчим пристроєм (реле).

Також передбачено можливість зворотного зв'язку: ESP32 може повідомляти стан реле назад на MQTT брокер, що забезпечує синхронізацію даних між пристроєм та застосунком.

3.2.3 Електронні компоненти

Для реалізації прототипу "розумної" розетки, що інтегрується до локальної мережі та керується дистанційно за допомогою жестів через мобільний застосунок, було обрано низку електронних компонентів, кожен з яких виконує специфічну функцію у загальній архітектурі пристрою.

Основним обчислювальним елементом конструкції є мікроконтролер ESP32 (який зображений на рисунку 3.5), що поєднує у собі потужний двоядерний процесор, достатню кількість цифрових і аналогових входів/виходів, а також модулі бездротового зв'язку Wi-Fi та Bluetooth. Наявність цих інтерфейсів дає змогу реалізувати як первинне налаштування пристрою (через Bluetooth), так і подальший обмін даними з MQTT-брокером через Wi-Fi, що є ключовим для організації хмарного контролю IoT-пристроїв.

Окрім ESP32, до схеми було включено електромеханічне реле, яке виконує безпосереднє замикання або розмикання електричного кола, керуючи подачею живлення до підключеного споживача. Також застосовано стабілізатор напруги, що забезпечує безпечне живлення логічних елементів, та світлодіодні індикатори для зручного відображення стану пристрою. Така комбінація компонентів забезпечує надійну, функціональну та енергоефективну роботу системи в умовах побутового використання.

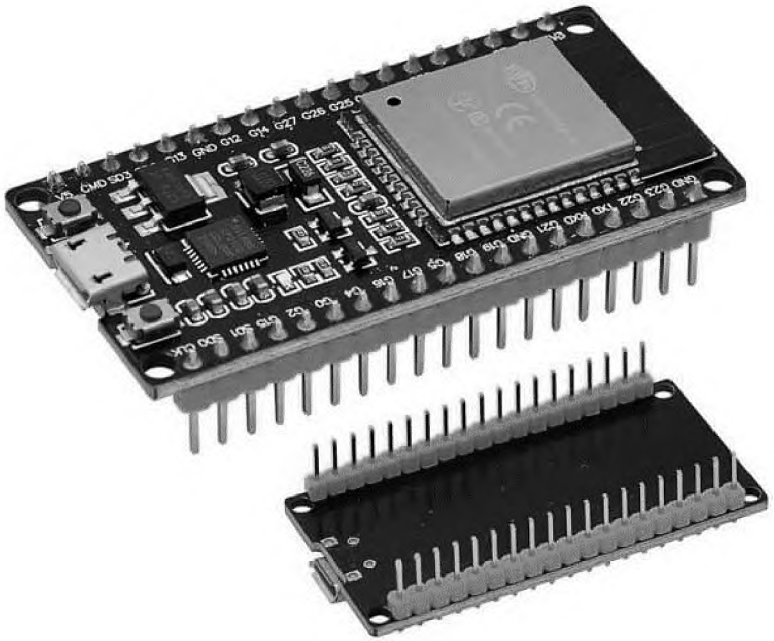

Рисунок 3.5 — Мікроконтролер ESP32

Оскільки мікроконтролер працює при напрузі 3.3–5 В постійного струму, виникає необхідність у зниженні та стабілізації напруги з побутової електромережі 220 В змінного струму. Для цієї мети використано блок живлення Hi-Link HLK-PM01 (який зображений на рисунку 3.6), що являє собою компактний імпульсний модуль, призначений для перетворення мережевої напруги на стабілізовані 5 В постійного струму з достатнім рівнем ізоляції, що забезпечує безпечну експлуатацію пристрою в побутових умовах.

Рисунок 3.6 — Блок живлення Hi-Link HLK-PM01

Функцію комутації електричного навантаження виконує електромагнітне реле (який зображений на рисунку 3.7), типове для низьковольтного керування високовольтними пристроями. У цьому випадку реле служить проміжним елементом між мікроконтролером та фазним проводом електричного кола.

Рисунок 3.7 — Електромагнітне реле

Коли ESP32 отримує команду через MQTT, на його вихід подається сигнал, який через транзистор NPN (наприклад, S8050який зображений на рисунку 3.8) активує реле, замикаючи або розмикаючи силовий контакт. Таким чином, здійснюється вмикання або вимикання живлення на кінцевому електропристрої, що підключений до розетки.

Щоб захистити транзистор та мікроконтролер від зворотної електрорушійної сили, яка виникає в момент вимкнення реле, в коло паралельно до його котушки включено діод типу 1N4007 (який зображений на рисунку 3.9). Це дозволяє уникнути пошкодження електроніки при інтенсивному перемиканні навантаження.

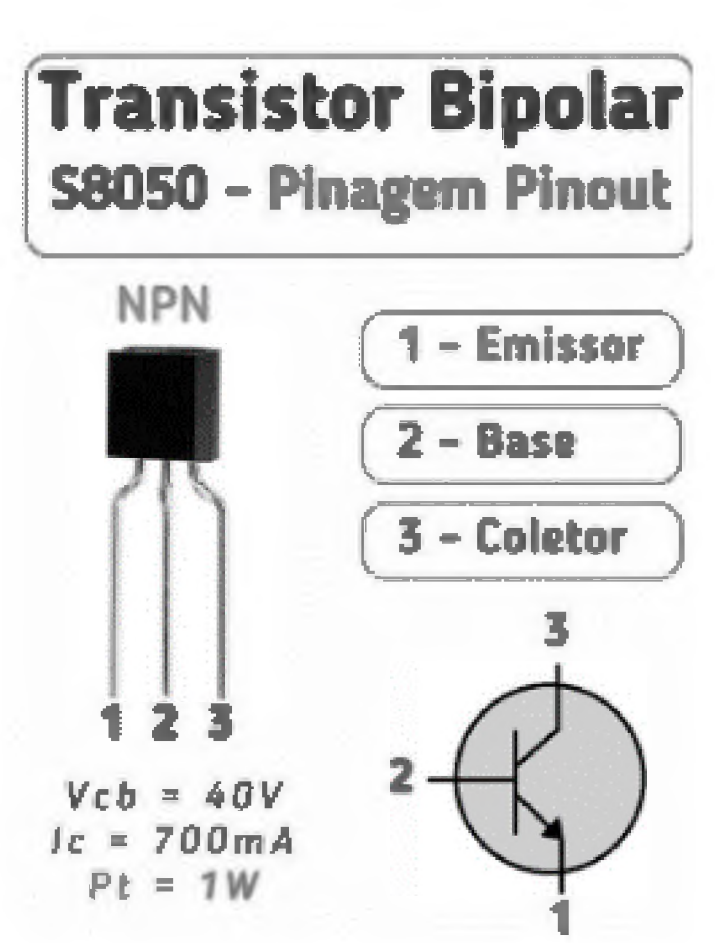

Рисунок 3.8 — Транзистор NPN

Рисунок 3.9 — Діод типу 1N4007

Для забезпечення візуального контролю за станом пристрою в схему додано світлодіодний індикатор, який сигналізує про активність пристрою (наприклад, про замкнене реле чи успішне з’єднання з мережею). Світлодіод живиться через обмежувальний резистор, який запобігає надмірному струму через напівпровідниковий компонент (який зображений на рисунку 3.10). Керування індикатором здійснюється за допомогою одного з цифрових виходів ESP32, що дозволяє програмно вмикати або вимикати світлодіод відповідно до поточного стану системи. Це спрощує діагностику пристрою користувачем та забезпечує додатковий зворотний зв’язок при взаємодії з IoT-системою.

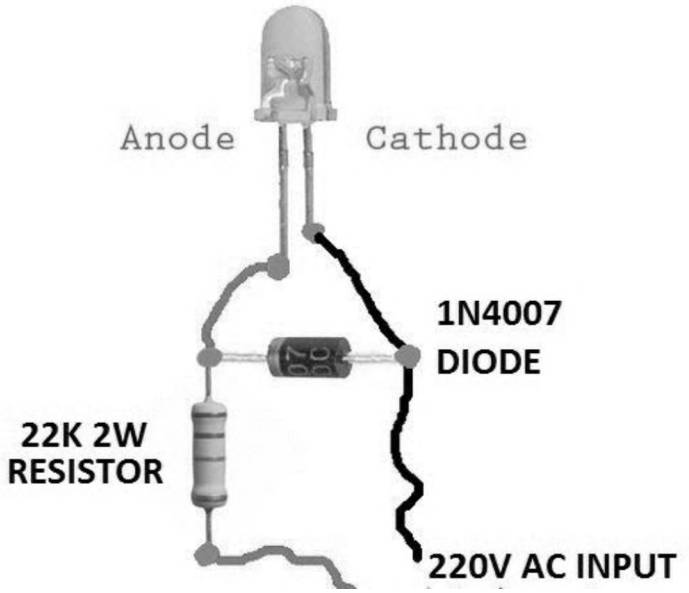

Рисунок 3.10 — Світлодіодний індикатор та резистор

З метою зручного монтажу провідників і забезпечення надійного електричного контакту використовуються гвинтові клемники. Вони дозволяють легко підключити силові дроти без пайки, що особливо важливо для високовольтної частини пристрою.

Всі компоненти змонтовані на макетній платі або друкованій платі (PCB) (яка зображена на рисунку 3.11), що забезпечує компактність, зручність технічного обслуговування та мінімізацію контактних помилок.

Для експлуатації пристрою в реальних умовах він розміщується в герметичному корпусі з термостійкого пластику (який зображений на рисунку 3.12), який запобігає випадковому доступу до високовольтних елементів та забезпечує належний рівень електробезпеки.

Також корпус має вентиляційні отвори, що сприяють ефективному відведенню тепла під час роботи реле та мікроконтролера. Це дозволяє уникнути перегріву компонентів та продовжує термін служби пристрою в умовах тривалої експлуатації.

Крім того, конструкція корпусу передбачає можливість його фіксації на стіні або в стандартному монтажному боксі, що полегшує встановлення пристрою у побутових умовах.

Рисунок 3.11 — Макетна плата

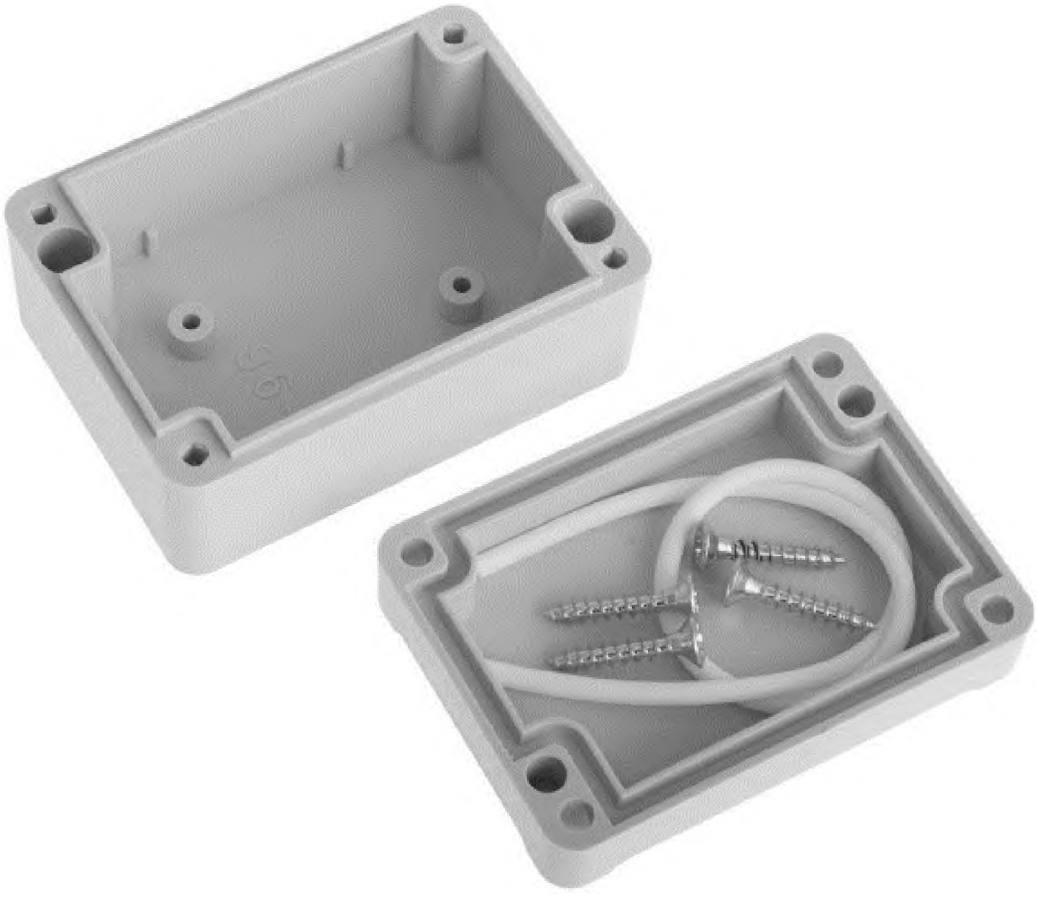

Рисунок 3.12 — Герметичний корпус з термостійкого пластику

Таким чином, запропонована конструкція є повноцінним IoT-рішенням, яке дозволяє керувати електроживленням через інтерфейс мобільного застосунку, використовуючи лише жести рук. При цьому пристрій поєднує в собі безпеку, функціональність та масштабованість, що відкриває перспективи для подальшого розвитку системи та інтеграції її у більш складні сценарії розумного дому.

Перелік компонентів для реалізації системи зображені в таблиці 3.1.

Таблиця 3.1 — Перелік компонентів для реалізації системи

№	Компонент	Опис
1	Мікроконтролер ESP32 DevKit	Основний контролер з вбудованими Wi-Fi та Bluetooth модулями. Має цифрові/аналогові входи-виходи.
2	Електромагнітне реле 5В (типу SRD-05VDC-SL-C)	Керований перемикач, який дозволяє замикати/розмикати силове коло з використанням сигналу низької напруги.
3	Блок живлення Hi-Link HLK-PM01 (220В -> 5В)	Імпульсний модуль живлення для перетворення змінної напруги 220 В на постійну 5 В. Компактний та ізольований.
4	Діод 1N4007	Використовується для захисту мікроконтролера від зворотного струму реле (встановлюється паралельно до котушки реле).
5	Резистори (1 кОм, 10 кОм)	Для обмеження струму або формування подільників напруги (наприклад, у кнопках або вхідних сигналах).
6	Світлодіод (LED)	Для індикації стану пристрою, наприклад, ввімкнено/вимкнено живлення.
7	Транзистор NPN (наприклад, 2N2222 або S8050)	Підсилювач для керування реле, коли мікроконтролер не може напряму подати достатній струм.
8	Клемник або гвинтові затискачі	Для підключення фазного проводу електромережі до схеми без пайки.
9	Друкована плата (PCB) або макетна плата	Основна платформа для збирання схеми. PCB рекомендується для надійності.
10	Корпус (бажано негорючий пластик)	Забезпечує безпеку та ізоляцію при роботі з напругою 220 В. Має отвори для вентиляції та доступу до розетки.
11	Проводи та роз'єми (Dupont/клеми)	Для з’єднання елементів між собою, зручні у монтажі та заміні компонентів.

3.2.4 Програмне забезпечення пристрою

Для забезпечення належної взаємодії між компонентами пристрою та зовнішнім програмним середовищем (мобільним застосунком і MQTT-брокером) була розроблена прошивка мікроконтролера ESP32. Основні функції цієї прошивки полягають у налаштуванні бездротових інтерфейсів, керуванні комутацією електроланцюга та обробці вхідних команд.

Під час початкової прошивки у постійну пам’ять мікроконтролера записується унікальний ідентифікатор пристрою — UUID (Universally Unique Identifier), який надалі використовується для ідентифікації девайса серед інших учасників IoT-мережі. Завдяки цьому кожне повідомлення, отримане через MQTT, може бути оброблене тільки тим пристроєм, якому воно призначене.

Прошивка реалізує два основних режими роботи пристрою: режим Bluetooth (початкове налаштування) та режим Wi-Fi (робоча експлуатація). Для зручності перемикання між цими режимами в коді реалізовано перелік (enum), у якому визначено відповідні стани. При першому ввімкненні пристрій автоматично переходить у Bluetooth-режим і очікує підключення з боку смартфона. Як тільки з'єднання встановлено, пристрій очікує на отримання конфігураційних даних: імені та пароля Wi-Fi-мережі, а також облікових даних і теми MQTT-брокера.

Дані надсилаються у форматі:

SsidName#SsidPassword#MqttLogin#MqttPassword#MqttTopic

Після прийому рядка конфігурації, мікроконтролер розбиває його на окремі частини за роздільником # для подальшої обробки. Для підключення до Wi-Fi-мережі використовується функція WiFi.begin(ssid, password) з бібліотеки WiFi. Якщо спроб підключення виявляється понад 10, вважається, що з’єднання не вдалося. У такому разі користувач отримує відповідне повідомлення про помилку через Bluetooth-з'єднання.

У разі успішного підключення до локальної мережі, пристрій намагається встановити з’єднання з MQTT-брокером за допомогою методів:

client.setServer(mqttUrl, mqttPort);
client.connect(mqttLogin, mqttPassword).

Аналогічно до попереднього етапу, після десяти невдалих спроб з'єднання користувач інформується про помилку.

У випадку вдалого проходження обох етапів (Wi-Fi та MQTT), мікроконтролер передає через Bluetooth повідомлення про успішне налаштування та переходить у робочий режим. При цьому Bluetooth-модуль вимикається для економії енергії та зниження ризиків стороннього втручання, а пристрій починає прослуховувати MQTT-тему, очікуючи на керуючі команди.

Вхідні MQTT-повідомлення очікуються у форматі:

uuid#команда

Де uuid — унікальний ідентифікатор пристрою, а команда — одна з двох можливих: "on" або "off". Після отримання повідомлення мікроконтролер порівнює UUID з власним. Якщо ідентифікатори збігаються, відбувається перевірка стану пристрою. Якщо поточний стан відповідає вхідній команді — жодних дій не здійснюється. У протилежному випадку виконується зміна стану шляхом подання або зняття сигналу на цифровому виході, до якого підключено електромагнітне реле.

Команда "on" активує реле, замикаючи електричний ланцюг, що дозволяє подати живлення на навантаження (наприклад, побутовий прилад).

Команда "off" розмикає реле, тим самим розриваючи ланцюг і припиняючи подачу напруги.

Цикл обробки повідомлень відбувається постійно, забезпечуючи в реальному часі можливість керування пристроєм через мобільний застосунок або інший інтерфейс, що підтримує публікацію повідомлень у відповідну тему MQTT-брокера.

Завдяки такому підходу вдалося реалізувати стабільну та гнучку архітектуру керування, яка відповідає сучасним принципам побудови IoT-систем.

Код програми наведено в додатку.

У випадку, якщо на пристрій надходить команда, що не відповідає UUID, вона ігнорується як неактуальна. Такий підхід забезпечує можливість одночасної роботи багатьох однакових пристроїв у межах однієї мережі без конфліктів, адже кожен з них "слухає" лише повідомлення, адресовані саме йому.

З метою надійності було реалізовано повторне підключення до MQTT-брокера в разі втрати зв’язку. У циклі loop() перевіряється статус з’єднання з брокером, і в разі його втрати виконується повторне підключення з використанням збережених облікових даних. Це дозволяє гарантувати безперервну роботу пристрою в умовах нестабільного інтернет-з'єднання.

Для зручності керування пристроєм користувачем зі смартфона, була реалізована проста структура команд та формат повідомлень. Усі повідомлення, що надходять з брокера, мають формат:

UUID#команда

де UUID — це унікальний ідентифікатор пристрою, а команда — або on, або off.

У разі подання команди on, контролер активує реле, замикаючи електричне коло, завдяки чому до споживача подається електроенергія. При отриманні команди off, реле розмикається, і споживач знеструмлюється. Таким чином, за допомогою простих MQTT-команд користувач може керувати подачею живлення на підключені до розетки електроприлади.

Варто зазначити, що дана реалізація дозволяє гнучко адаптувати систему до потреб користувача: за потреби можливо додати нові типи команд, реалізувати двосторонню комунікацію (тобто повідомлення від пристрою на MQTT-брокер про свій стан), або розширити функціонал за допомогою датчиків (наприклад, моніторингу споживаної потужності, температури тощо).

Крім того, така архітектура відкриває широкі можливості для інтеграції з системами "розумного дому", платформами автоматизації (наприклад, Home Assistant), або мобільними застосунками для віддаленого контролю.

3.3 Візуалізація та демонстрація роботи ключових елементів системи

Представлено візуалізацію та демонстрацію основних елементів, що складають розроблену систему, зокрема зображення "розумної" розетки та інтерфейсу мобільного додатку для керування IoT пристроями. Ключовими елементами є як апаратні компоненти, так і програмне забезпечення, що забезпечує взаємодію з ними, включаючи управління пристроєм за допомогою жестів через мобільний додаток. Візуалізація дає можливість детальніше уявити, як система працює та які інтерфейси використовуються для зручності користувача.

На рисунку 3.13 представлений зовнішній вигляд "розумної" розетки. Пристрій виконаний у стандартному корпусі, що дозволяє легко інтегрувати його в будь-яке стандартне електричне обладнання, що використовується в побуті. Розетка має кілька індикаторів, які вказують на поточний стан пристрою: підключення до Wi-Fi мережі, активність в мережі MQTT, а також стан самого електричного ланцюга. Такий дизайн дозволяє користувачу одразу побачити, чи працює розетка, і в разі потреби здійснити коригування чи налаштування.

Інтерфейс Android застосунку для управління "розумною" розеткою представлений на рисунку 3.14. Застосунок має простий та інтуїтивно зрозумілий інтерфейс, який включає кілька ключових елементів: основний екран для вибору пристрою з переліку доступних IoT девайсів, вбудовану камеру для зчитування жестів, а також меню для налаштувань пристрою. У цьому інтерфейсі також розміщена пам’ятка, яка містить докладний опис доступних жестів для управління кожним підключеним IoT пристроєм.

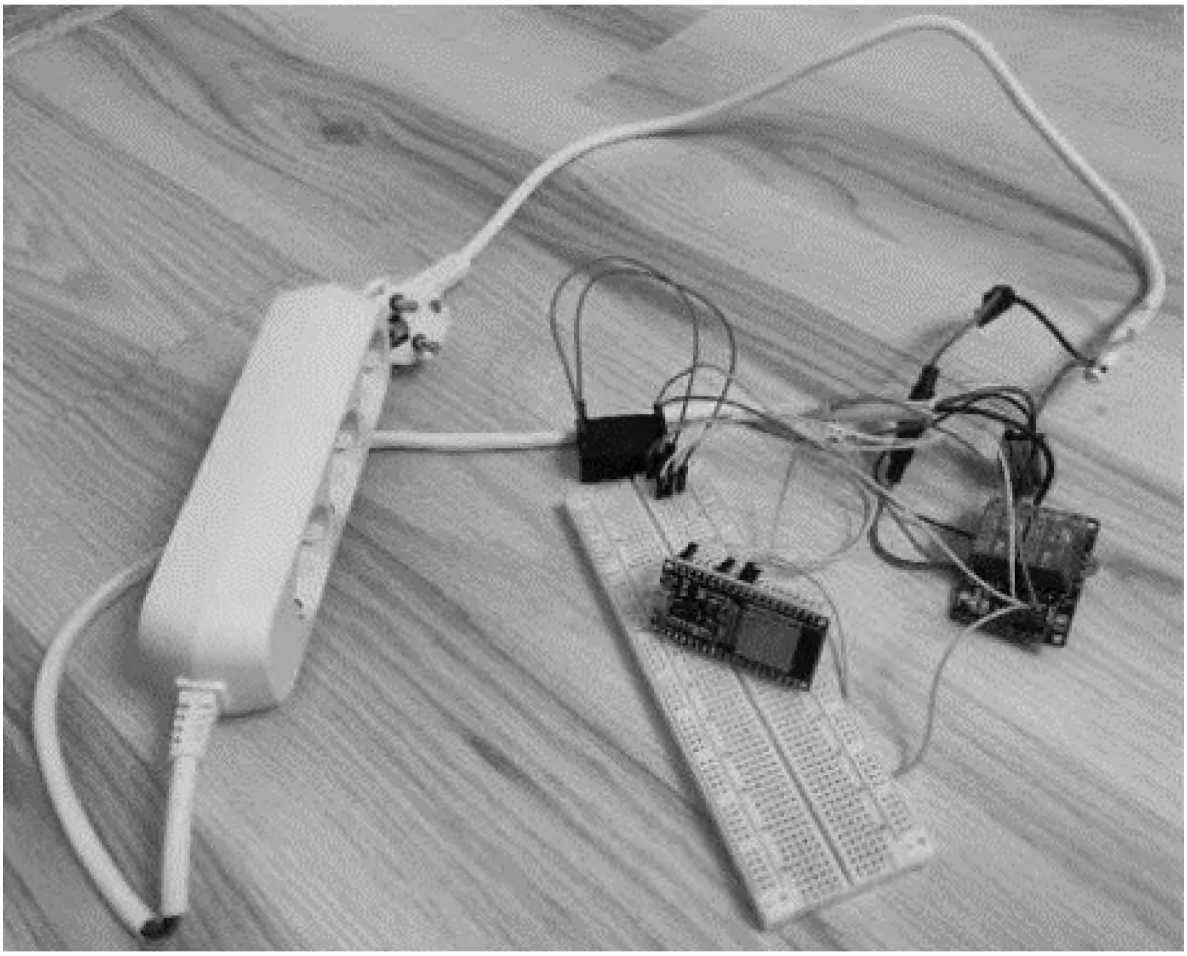

Рисунок 3.13 — Зовнішній вигляд макету системи

Користувач може, використовуючи камеру та алгоритми комп'ютерного зору, передавати команди пристрою через відповідні жести рук. Така технологія не лише забезпечує зручне, але й безконтактне управління пристроєм, що може бути корисним у ситуаціях, коли необхідно уникати фізичного контакту з пристроєм або підвищити зручність керування.

Особливістю інтерфейсу є адаптивність до різних типів пристроїв, що дозволяє контролювати декілька девайсів одночасно, змінюючи їхні налаштування і стани без необхідності вручну перемикати екран або програму. Це забезпечує користувачам максимальний комфорт під час взаємодії з IoT пристроями.

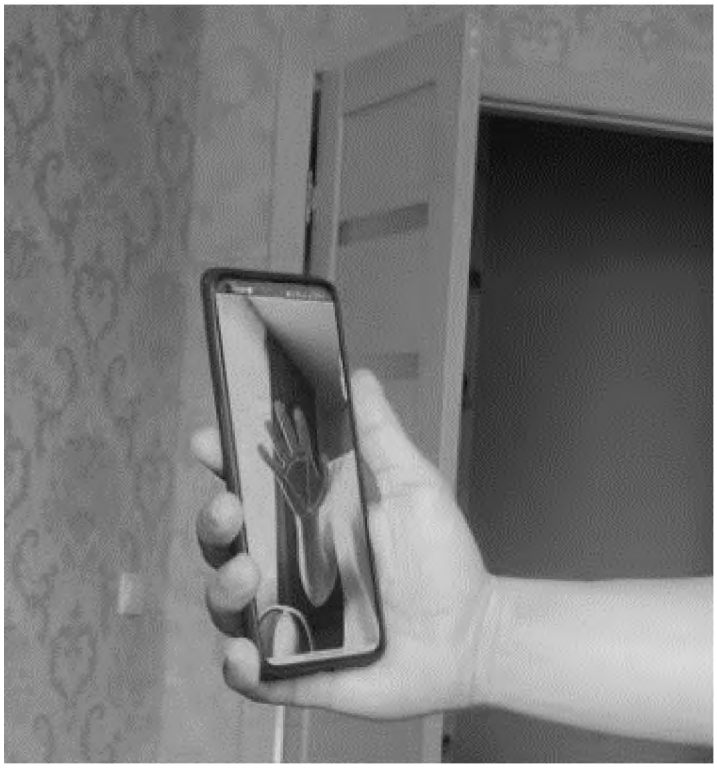

Рисунок 3.14 — Інтерфейс Android застосунку

Візуалізація в цілому дозволяє побачити ефективність використання мобільного додатку для управління "розумною" розеткою через інтерфейс, що забезпечує можливість здійснювати команди на основі зчитування жестів рук.

3.4 Аналіз продуктивності та ефективності реалізованого рішення

Оцінка продуктивності та ефективності розробленого програмного забезпечення є важливим етапом для визначення його здатності виконувати основні функції та задовольняти вимоги користувачів. У контексті даної роботи основною метою є забезпечення стабільного та точного розпізнавання жестів для ефективного управління IoT пристроями в умовах розумного будинку. Тому однією з ключових метрик для оцінки успіху розробки є точність розпізнавання жестів, адже система повинна реагувати на певні жести з максимальною точністю і мінімумом помилок.

Для вимірювання точності розпізнавання жестів було проведено експериментальне дослідження. П'ять учасників експерименту виконували сто спроб керування "розумною" розеткою через Android додаток, використовуючи

різні жести рук. У процесі експерименту були зафіксовані не лише кількість успішно розпізнаних жестів, а й помилки, пов'язані з неправильним або неточним визначенням жестів.

Експеримент було проведено таким чином, щоб врахувати можливі варіації в умовах виконання жестів, включаючи різні положення рук, швидкість виконання та освітлення. Кожен учасник мав можливість виконати жести в різних варіантах, щоб перевірити, наскільки система здатна адаптуватися до змінних умов.

Загалом, результати експерименту, представлені в таблиці 3.2, показують, що точність розпізнавання жестів знаходиться на достатньо високому рівні, що дозволяє ефективно використовувати систему для управління IoT пристроями в реальному житті. Однак важливо зазначити, що деякі помилки в розпізнаванні жестів все ж мали місце, що вказує на необхідність подальшої оптимізації алгоритмів розпізнавання для досягнення ще більш високої точності.

З наведених результатів видно, що середній відсоток успішно розпізнаних жестів варіюється в межах 94-98%. Ці показники є дуже хорошими, оскільки в реальних умовах часто виникають фактори, які можуть впливати на точність, зокрема, освітлення, швидкість виконання жестів, а також точність рухів рук. Зокрема, учасники з більш стабільними умовами (відсутність сильних змін освітлення, точне виконання жестів) показали кращі результати, що підтверджує важливість зовнішніх умов для точності розпізнавання.

Таблиця 3.2 — Підсумки проведеного експерименту розпізнавання жестів

№ особи за порядком	Кількість проведених експериментів	Відсоток правильно розпізнаних жестів
1.	100	92
2.	100	87
3.	100	93
4.	100	92
5.	100	90
Середній відсоток точності розпізнавання		**90,8**

Відзначаючи, що деякі користувачі стикалися з помилками при виконанні певних жестів (наприклад, при використанні швидких або не зовсім точних рухів рук), можна стверджувати, що система має потенціал для подальшого вдосконалення. У разі впровадження додаткових механізмів корекції (наприклад, використання методів машинного навчання для адаптації до індивідуальних особливостей користувачів або покращення алгоритмів фільтрації шумів), точність розпізнавання може бути значно підвищена.

Загалом, отримані результати свідчать про те, що система здатна забезпечити достатньо високу точність розпізнавання жестів, щоб бути корисною для керування розумними пристроями в реальному житті. Технологія виявилась життєздатною та ефективною для впровадження в систему IoT управління, однак для досягнення максимальної ефективності необхідно врахувати фактори навколишнього середовища та постійно працювати над оптимізацією алгоритмів для різних користувачів та умов.

ВИСНОВКИ

У межах кваліфікаційної роботи було реалізовано комп'ютерну систему для керування IoT-пристроями, яка використовує мобільний застосунок на базі Android та технології розпізнавання жестів. Метою дослідження було створення розширюваної платформи, здатної забезпечити гнучке управління різноманітними IoT-компонентами, що успішно досягнуто в результаті виконаних завдань.

У ході роботи:

– проведено аналіз існуючих підходів до управління IoT-пристроями та визначено їх переваги й обмеження;

– розроблено масштабовану архітектуру системи, яка дозволяє інтегрувати нові типи пристроїв без істотних змін у структурі;
– досліджено методи зчитування жестів із відеопотоку та впроваджено відповідні алгоритми у мобільний застосунок;

– створено Android-клієнт з підтримкою взаємодії користувача з IoT-пристроями через зручний графічний інтерфейс і жести;
– виготовлено та протестовано власний IoT-пристрій із відкритим інтерфейсом для приймання команд;

– здійснено повноцінне тестування системи, яке засвідчило її стабільність, адаптивність і можливість практичного застосування.

Таким чином, запропонована комп'ютерна система керування IoT-пристроями є перспективною для використання як у домашніх умовах, так і в комерційних або промислових середовищах. Вона слугує платформою для подальших досліджень у галузі розумних середовищ, зручних інтерфейсів керування та побудови персоналізованих IoT-рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fibaro. Swipe [Електронний ресурс]. – Режим доступу: https://www.fibaro.com/ru/products/swipe/ – Дата звернення: 03.05.2025.

2. Leap Motion [Електронний ресурс]. – Режим доступу: https://vr-store.com.ua/ua/leap-motion-detail – Дата звернення: 03.05.2025.

3. Mediapipe [Електронний ресурс]. – Режим доступу: https://developers.google.com/ – Дата звернення: 03.05.2025.

4. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/1704.04861 – Дата звернення: 03.05.2025.

5. SSD object detection: Single Shot MultiBox Detector for real-time processing [Електронний ресурс]. – Режим доступу: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multiboxdetector-for-real-time-processing-9bd8deac0e06 – Дата звернення: 03.05.2025.

6. Facial Landmarks Estimation [Електронний ресурс]. – Режим доступу: https://docs.nvidia.com/tao/tao-toolkit/text/facial_landmarks_estimation/facial_landmarks_estimation.html – Дата звернення: 03.05.2025.

7. Що таке “впровадження залежностей”? [Електронний ресурс]. – Режим доступу: https://doc.nette.org/uk/dependency-injection/introduction – Дата звернення: 03.05.2025.

8. Choosing Android Architectures: MVC, MVP, MVVM, Clean Architecture, and MVI [Електронний ресурс]. – Режим доступу: https://medium.com/@KodeFlap/choosing-android-architectures-mvc-mvp-mvvm-clean-architecture-and-mvi-8ad2a43f7f9b – Дата звернення: 03.05.2025.

9. Скетчі для Arduino [Електронний ресурс]. – Режим доступу: https://arduino-ide.com/faq/123-sketchi-dlja-arduino.html – Дата звернення: 03.05.2025.

10. Activity [Електронний ресурс]. – Режим доступу: https://developer.android.com/reference/android/app/Activity – Дата звернення: 03.05.2025.

11. Fragments [Електронний ресурс]. – Режим доступу: https://developer.android.com/guide/fragments – Дата звернення: 03.05.2025.

12. Імперативне та декларативне програмування і чим вони відрізняються [Електронний ресурс]. – Режим доступу: https://foxminded.ua/imperatyvne-ta-deklaratyvne-prohramuvannia/ – Дата звернення: 03.05.2025.

13. Декларативний підхід при створенні мультиплатформних додатків [Електронний ресурс]. – Режим доступу: https://gb.ru/blog/imperativnoe-programmirovanie/https://journals.snu.edu.ua/index.php/VisnikSNU/article/view/577 – Дата звернення: 03.05.2025.

14. Імперативне програмування [Електронний ресурс]. – Режим доступу: http://ruslan.rv.ua/python-essential/paradigmas/imperative.html – Дата звернення: 03.05.2025.

15. HiveMQ MQTT Client [Електронний ресурс]. – Режим доступу: https://hivemq.github.io/hivemq-mqtt-client/docs/installation/android/ – Дата звернення: 03.05.2025.

16. ESP32 [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/ESP32 – Дата звернення: 03.05.2025.

17. HLK-PM01 [Електронний ресурс]. – Режим доступу: https://www.mini-tech.com.ua/ua/blok-pitaniya-5v-600ma-kompakt – Дата звернення: 03.05.2025.

18. Бедрій І. Я., Нечай В. Я. Безпека життєдіяльності : навч. посіб. – Львів : Манголія 2006, 2007. – 499 с.

19. Желібо Є. П., Заверуха Н. М., Зацарний В. В. Безпека життєдіяльності : навч. посіб. – Київ : Каравела, 2004. – 328 с.

20. Зеркалов Д. В. Безпека життєдіяльності : навч. посіб. – Київ : Основа, 2011. – 526 с.

21. Осухівська Г. М., Тиш Є. В., Луцик Н. С., Паламар А. М. Методичні вказівки до виконання кваліфікаційних робіт здобувачів першого

(бакалаврського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. – Тернопіль : ТНТУ, 2022. – 28 с.

22. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп'ютерні мережі. Кн. 1 : навч. посіб. – Львів : Магнолія 2006, 2013. – 256 с.

23. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп'ютерні мережі. Кн. 2 : навч. посіб. – Львів : Магнолія 2006, 2014. – 312 с.

24. Лупенко С. А., Пасічник В. В., Тиш Є. В. Комп'ютерна логіка. – Львів : Магнолія 2006, 2015. – 354 с.

25. Паламар М. І., Стрембіцький М. О., Паламар А. М. Проектування комп'ютеризованих вимірювальних систем і комплексів : навч. посіб. – Тернопіль : ТНТУ, 2019. – 150 с.

26. Yatsyshyn V., Pastukh O., Palamar A., Zharovskyy R. Technology of relational database management systems performance evaluation during computer systems design // Scientific Journal of TNTU. – 2023. – Vol. 109, No 1. – P. 54–65.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф.

_____О. Д. Азаров

“__” _____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи
«Комп'ютерна система керування IoT пристроями»
08-54.БКР.010.00.000 ТЗ

Науковий керівник к.т.н., доц. каф. ОТ

_____ Колесник І.С.

Студент групи 1КІ-23мс

_____ Куляс О.П.

Вінниця 2025

1 Найменування та область застосування

Комп'ютерна система керування IoT пристроями.

2 Основи для розробки

Необхідність реалізації комп'ютерної системи керування IoT пристроями

3 Мета та призначення розробки

Метою розробки є розробка комп’ютерної системи для централізованого управління IoT-пристроями із застосуванням жестового управління за допомогою смартфона з ОС Android та створення власного IoT-пристрою для інтеграції в систему.

4 Етапи БКР та очікувані результати

Робота виконується у вісім етапів, що наведені в таблиці 4.1.

Таблиця 4.1 — Етапи виконання роботи

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Постановка задачі роботи		
2	Огляд та аналіз галузі дослідження,		
3	Теоретичні аспекти розробки системи,		
4	Практична реалізація системи		
5	Реалізація апаратної частини IoT-рішення		
6	Розробка програмного забезпечення пристрою		
7	Оформлення пояснювальної записки та ілюстративного матеріалу		
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків		

5 Матеріали, що подаються до захисту БКР

Пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відзив наукового керівника, рецензія опонента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

6 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

7 Вимоги до оформлювання та порядок виконання БКР.

7.1 При оформлювання БДР використовуються:

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

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

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

— Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

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

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:
Комп'ютерна система керування IoT пристроями

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР,МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра,факультет)

Показники звіту подібності Strike Plagiarism

Оригінальність 99%

Схожість 1%

Аналіз звіту подібності (відмітити потрібне):

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

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

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

Особа,відповідальназаперевірку _____ Захарченко С.М.
(підпис) (прізвище,ініціали)

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

Автор роботи _____ Куляс О.П,
(підпис) (прізвище,ініціали)

Керівник роботи _____ Колесник І.С.

ДОДАТОК В

Графічна частина

КОМП'ЮТЕРНА СИСТЕМА КЕРУВАННЯ ІОТ ПРИСТРОЯМИ

Послідовність взаємодії між мобільним клієнтом, MQTT брокером та IoT-пристроєм

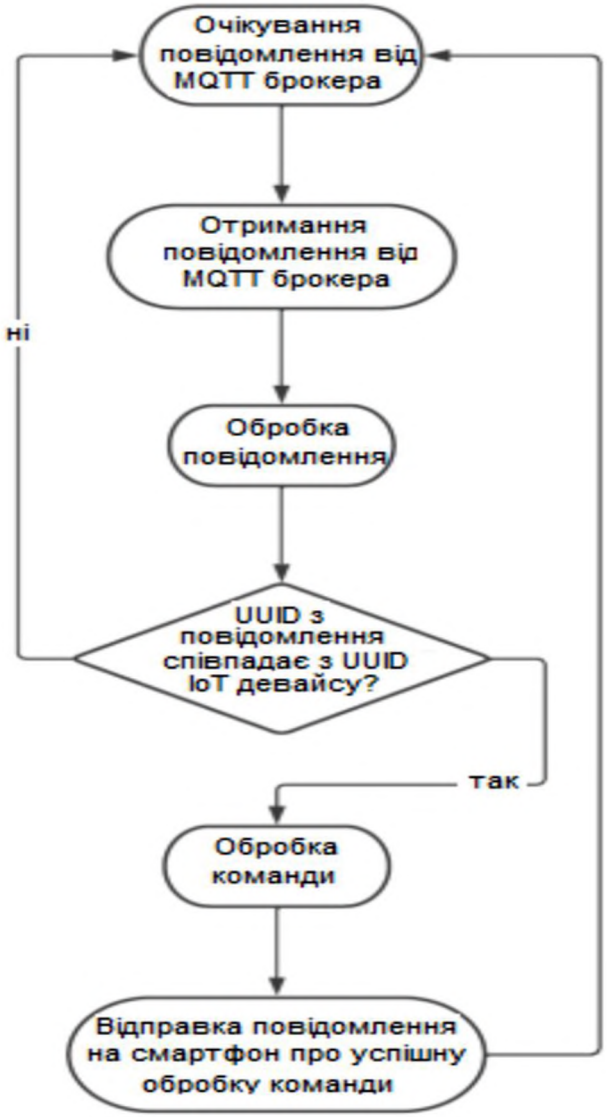

Рисунок В1 — Послідовність взаємодії між мобільним клієнтом, MQTT брокером та IoT-пристроєм

ДОДАТОК Г

Принцип роботи IoT пристроя

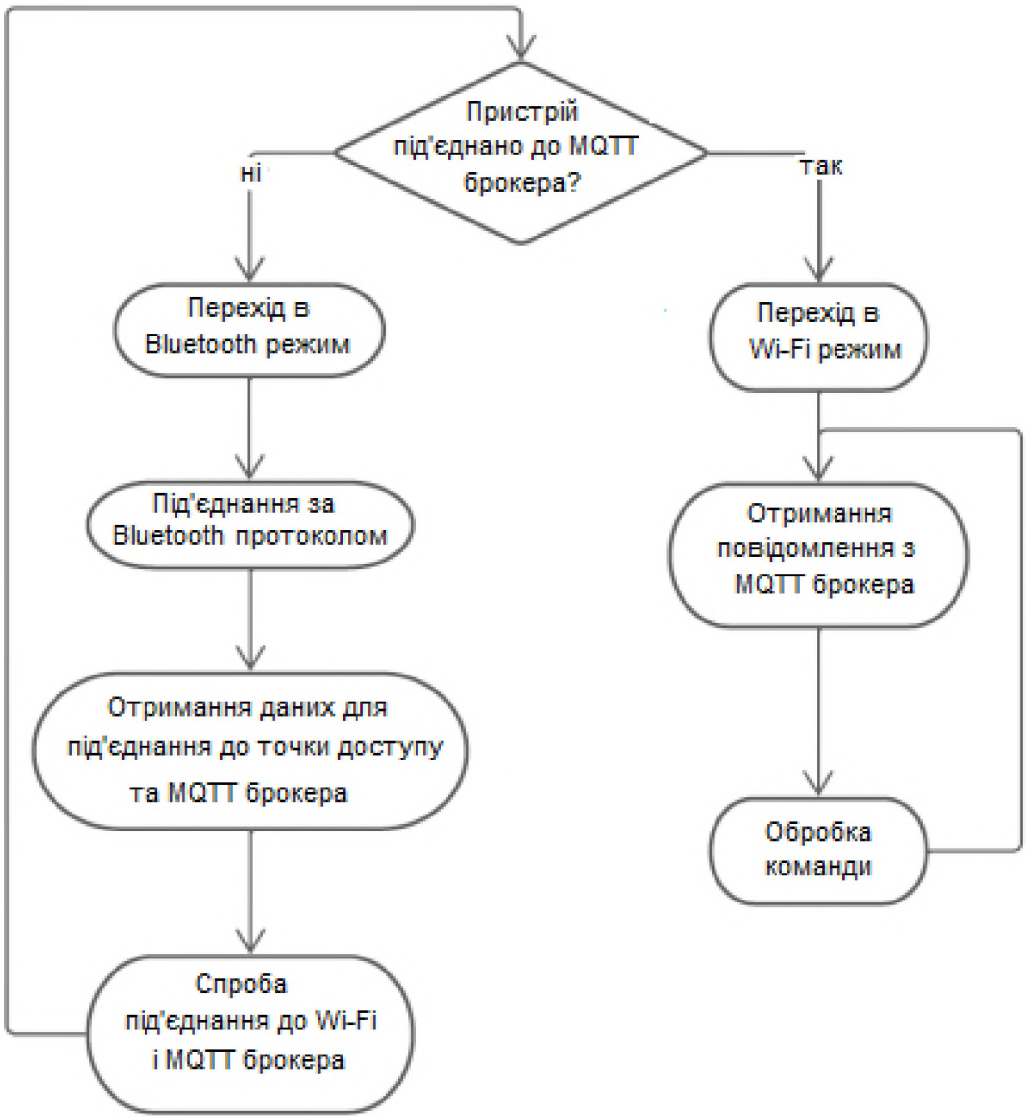

Рисунок Г1 — Принцип роботи IoT пристроя

ДОДАТОК Д

Код програми

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

#define RELAY_PIN 5
BluetoothSerial SerialBT;
WiFiClient espClient;
PubSubClient client(espClient);

String deviceUUID = "f47ac10b-58cc-4372-a567-0e02b2c3d479"; // Унікальний
UUID
String mqttTopic = "";
bool relayState = false;

enum DeviceMode { BLUETOOTH, WIFI };
DeviceMode currentMode = BLUETOOTH;

void setup() {
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);

  Serial.begin(115200);
  SerialBT.begin("SmartSocket");

  while (currentMode == BLUETOOTH) {
    if (SerialBT.available()) {
      String configData = SerialBT.readStringUntil('\n');
      configureDevice(configData);
    }
  }
}

void configureDevice(String data) {
  int splitIndex = 0;
  String parts[5];
  for (int i = 0; i < 5; i++) {
    splitIndex = data.indexOf('#');
    parts[i] = data.substring(0, splitIndex);
    data = data.substring(splitIndex + 1);
  }
```

```
  const char* ssid = parts[0].c_str();
  const char* password = parts[1].c_str();
  const char* mqttUser = parts[2].c_str();
  const char* mqttPass = parts[3].c_str();
  mqttTopic = parts[4];

  WiFi.begin(ssid, password);
  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 10) {
    delay(1000);
    attempts++;
  }

  if (WiFi.status() != WL_CONNECTED) {
    SerialBT.println("WiFi_ERROR");
    return;
  }

  client.setServer("mqtt.example.com", 1883);
  attempts = 0;
  while (!client.connect("ESP32Client", mqttUser, mqttPass) && attempts < 10) {
    delay(1000);
    attempts++;
  }

  if (!client.connected()) {
    SerialBT.println("MQTT_ERROR");
    return;
  }

  SerialBT.println("CONNECTED");
  SerialBT.end();
  currentMode = WIFI;

  client.setCallback(mqttCallback);
  client.subscribe(mqttTopic.c_str());
}

void loop() {
  if (currentMode == WIFI) {
    if (!client.connected()) {
      // Повторне підключення при втраті з'єднання
      return;
    }
```

```
    client.loop();
  }
}

void mqttCallback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (unsigned int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  if (message.startsWith(deviceUUID)) {
    int delimiterIndex = message.indexOf('#');
    String command = message.substring(delimiterIndex + 1);

    if (command == "on" && !relayState) {
      digitalWrite(RELAY_PIN, HIGH);
      relayState = true;
    } else if (command == "off" && relayState) {
      digitalWrite(RELAY_PIN, LOW);
      relayState = false;
    }
  }
}
```

ДОДАТОК Є

Процес ініціалізації IoT пристрою

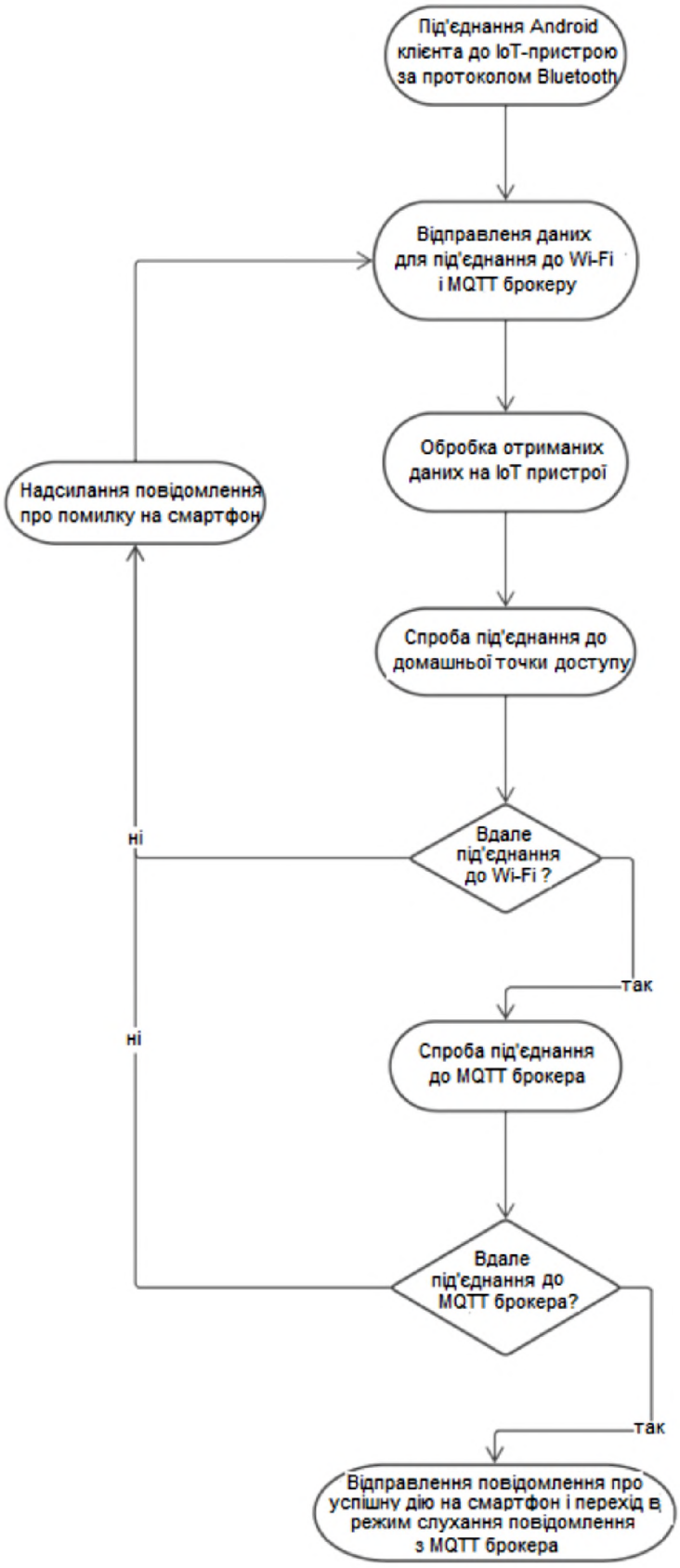

Рисунок Є1 — Процес ініціалізації IoT пристрою