МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ КЕРУВАННЯ ПРИСТРОЯМИ ІНТЕРНЕТУ РЕЧЕЙ»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
І.Р. Кос
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н., Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н., Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Косу Іллі Романовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка мобільного додатку для керування пристроями інтернету речей

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 09.01.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	29.01.2024	
2.	Збір вихідних даних.	15.02.2024	
3.	Обробка методичних та теоретичних	18.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	25.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2024	

Студент _____ І.Р. Кос
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка мобільного додатку для керування пристроями інтернету речей»: 44 с., 27 рис., 2 табл., 8 джерел, 1 додаток.

АНДРОЇД, ІНТЕРНЕТ РЕЧЕЙ, КОТЛІН, ЛАМПОЧКА, МОБІЛЬНИЙ ДОДАТОК, ОСВІТЛЕННЯ.

Об’єкт дослідження – плата ESP32.

Мета роботи: дослідити ринок сучасних мобільних додатків для керування освітленням, розробити мобільний додаток для керування пристроями інтернету речей.

Метод дослідження – методи програмної інженерії.

У цьому дослідженні буде проведено аналіз сучасних мобільних додатків для віддаленого керування освітленням, а також представлено огляд особливостей розробленої IoT системи, яка використовує платформу NodeMCU та мікроконтролер ESP32. Окрім того, описано програмну частину мобільного додатку для керування освітленням, що включає використання мови програмування Kotlin, елементи Arduino в апаратній частині та реалізацію сервера на Node.js для тестування. Дослідження завершується реалізацією та тестуванням мобільного додатка.

SUMMARY

Bachelor’s qualifying paper «Development of a Mobile Application for Internet of Things Devices Management»: 44 pages, 27 figures, 2 tables, 8 references, 1 supplement.

ANDROID, INTERNET OF THINGS, KOTLIN, LIGHT BULB, MOBILE APPLICATION, ROOM LIGHTING.

The object of the study is ESP32 development board.

The aim of the study is to research the market of modern mobile applications for lighting control, develop a mobile application for controlling Internet of Things devices.

The methods of research are software engineering methods.

This study will analyze modern mobile applications for remote lighting control and provide an overview of the features of the developed IoT system that uses the NodeMCU platform and the ESP32 microcontroller. In addition, the software part of the mobile application for lighting control is described, including the use of the Kotlin programming language, Arduino elements in the hardware, and the implementation of a Node.js server for testing. The study concludes with the implementation and testing of the mobile application.

ЗМІСТ

ВСТУП

Зважаючи на стрімкий розвиток технологічного прогресу, існує загальноприйнята потреба в зручному та ефективному управлінні пристроями через мобільні додатки. В наш час, коли смартфони є невіддільною частиною нашого побуту, стає очевидним, що мобільні додатки можуть стати ключовим інструментом для зручного та ефективного управління цими пристроями.

Ця кваліфікаційна робота зосереджується навколо огляду сучасних технологій інтернет речей для керування освітленням та створення аналога реальним мобільним додаткам з використанням мікропроцесора ESP-32 на пару з мовою програмування Kotlin в інтегрованому середовищі розробки Android Studio.

Мобільні додатки для керування пристроями Інтернету речей не лише роблять життя зручнішим та ефективнішим, але й відкривають нові горизонти для взаємодії між людьми та технологіями. Ці додатки стають мостом між віртуальним та реальним світом, де можливості обміну інформацією між різними пристроями стають не тільки реальністю, але й легкістю.

Однією з ключових функцій мобільного додатку для керування пристроями IoT є можливість керувати пристроями віддалено. Це дозволяє юзерові контролювати свої пристрої навіть тоді, коли вони не знаходяться вдома. Наприклад, користувач може перемикати систему опалення або світло через мобільний додаток, щоб зекономити енергію, створити ефект присутності вдома.

Отже, метою цієї кваліфікаційної роботи є розробка мобільного додатку для керування пристроями інтернету речей. Для досягнення мети передбачаються такі цілі:

– проаналізувати наявні мобільні додатки для керування освітленням пристроями інтернету речей;
– розробити дизайн інтерфейсу користувача мобільного додатку;
– забезпечити можливість віддаленого керування пристроями інтернету речей;
– провести тестування функцій додатка.

1 ПОРІВНЯЛЬНИЙ АНАЛІЗ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ ВІДДАЛЕНОГО КЕРУВАННЯ ОСВІТЛЕННЯМ

З розвитком сучасних технологій віддалене керування освітленням стало легким і зручним завдяки мобільним додаткам. Вибір правильного додатка для керування освітленням може бути складним завданням, оскільки на ринку існує велика кількість пропозицій. У цьому порівняльному аналізі розглянуто популярні мобільні додатки для віддаленого керування освітленням.

Philips Hue (див. рис. 1.1) [1].

Додаток є частиною однойменної екосистеми освітлення Philips Hue, яка дозволяє керувати освітленням через мобільний додаток. За допомогою додатка користувачі можуть перемикати та регулювати яскравість світла в будь-який час та з будь-якої точки світу. Крім того, вони можуть створювати розклади освітлення, змінювати колір світла, налаштовувати режими освітлення для різних ситуацій та інтегрувати свої розумні лампи з іншими смартпристроями у домашньому середовищі.

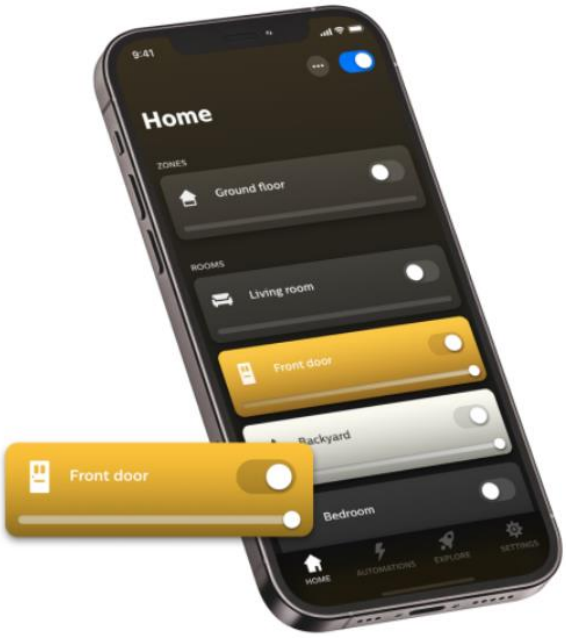

Рисунок 1.1 – Додаток Phillips Hue

Інтерфейс користувача:

- сучасний дизайн, доречно використані кольори на різних сторінках;
- основні функції шукаються легко, але більш специфічні знайти вже важко.

Функціональність:

- регуляція різних типів пристроїв освітлення, можливість зміни яскравості у відсотковому відношенні, що дозволяє точно підібрати потрібне значення;
- встановлення проміжку часу для функціонування освітлення;
- вбудована імітація присутності;
- відтворення звуку у вигляді освітлення.

Сумісність та розширення:

- малий вибір однотипних білих розумних ламп, пультів та аксесуарів, що призводить до неможливості дотримання будь-якого дизайну, окрім мінімалізму;
- неможливість під’єднання будь-яких пристроїв окрім пристроїв лінійки Philips Hue;
- має можливість співпрацювати з іншими додатками лінійки Philips Hue.

Підтримка та оновлення:

- регулярні оновлення, розробники та представники часто відповідають користувачам на відгуки у соц. мережах та крамницях застосунків.

Переваги:

- більшою мірою привабливий та зручний інтерфейс;
- достатньо функцій для аматорського керування пристроями освітлення;
- інтеграція з іншими додатками лінійки Philips hue.

Недоліки:

- дуже висока ціна для всіх продуктів, хоча додаток є безплатний;
- неможлива інтеграція з будь-якими пристроями поза своєї лінійки;
- хоча застосунок і має великий функціонал, але деякі можливості недоступні для користувача;

– відсутня можливість протестувати застосунок не маючи приладів для приєднання.

LIFX (див. рис. 1.2) [2].

Мобільний додаток для керування освітленням з можливістю гнучкої регуляції.

Рисунок 1.2 – Додаток LIFX

Інтерфейс користувача:

– сучасний дизайн і зручна навігація;
– легко доступні основні функції;
– зручне колесо за допомогою якого змінюється колір та відтінок лампи.

Функціональність:

– відображення шістнадцяти мільйонів кольорів;
– одночасне керування до 50 ламп;
– регуляція інтенсивності та теплоти освітлення;

– створення розкладів освітлення та налаштування сценаріїв;
– вбудована імітація присутності;
– можливість візуалізації музики та сповіщень.

Сумісність та розширення:

– широкий вибір розумних ламп та аксесуарів;
– інтеграція з іншими платформами домашньої авторизації, такими як: Google Home, Xfinity home, Razer, Amazon Alexa, Abode, IFTTT [2].

Підтримка та оновлення:

– за відгуками користувачів деякі оновлення спричиняють незадоволення користуванням застосунку.

Переваги:

– привабливий та зручний інтерфейс;
– достатньо функцій для аматорського керування пристроями освітлення;
– інтеграція з іншими платформами домашньої авторизації;
– освітлення може рухатися в такт музиці, імітувати тепле мерехтіння свічки;
– можливість зберігати сцени;
– наявна технологія поліхромії;
– доступні ціни.

Недоліки:

– велика кількість проблем, пов’язаних зі складнощами або неможливістю підключення ламп до додатка;
– іноді потребує мануальне перемикання.

Mi Home (див. рис. 1.3) [3].

Ключовий компонент екосистеми розумного освітлення від компанії Xiaomi.

Інтерфейс користувача:

– комплексний, але інтуїтивно зрозумілий дизайн.

Функціональність:

– додавання нових пристроїв через Wi-Fi, Bluetooth, QR-код тощо;

– створення сценаріїв для автоматичного виконання завдань на основі умов;
– отримання сповіщень про стан пристроїв у реальному часі;
– моніторинг енергоспоживання і роботи пристроїв;
– присутня синхронізація та резервне копіювання.

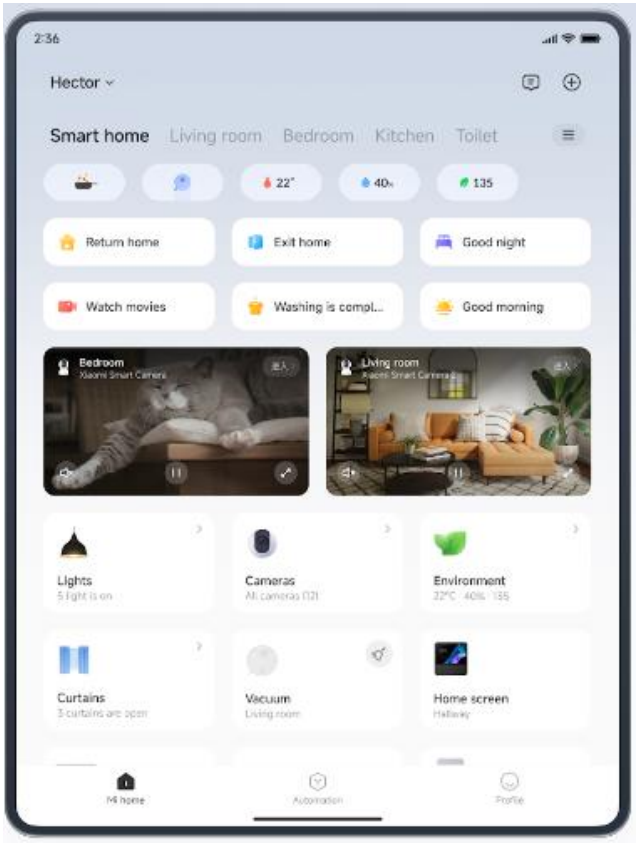

Рисунок 1.3 – Додаток Mi Home

Сумісність та розширення:

– дуже великий асортимент продукції починаючи з приладів для забезпечення безпеки, закінчуючи розумними лампочками, світильниками, ліхтарями;
– інтегровані голосові асистенти: Google Assistant, Amazon Alexa та Siri через HomeKit;
– присутня платформа розумного будинку IFTTT для створення складних сценаріїв автоматизації;
– інтеграція з сервісами погоди.

Підтримка та оновлення:

– регулярні оновлення програмного забезпечення;
– цілодобова служба підтримки.

Переваги:

– широка сумісність з різними пристроями, підтримка великої кількості пристроїв різних категорій;
– простий та зручний користувацький інтерфейс, який дозволяє легко додавати та керувати пристроями;
– інтеграція з іншими платформами, підтримка голосових асистентів.

Недоліки:

– вибір пристроїв у компанії Xiaomi дуже великий, але переважна більшість приладів від сторонніх виробників, з такими самими функціональними можливостями, є несумісними з додатком;
– у деяких моделей є проблеми з підключенням до додатка або просто нестабільна робота;
– має обмежену підтримку мов та регіонів, деякі функції там не доступні;
– залежить від хмарних сервісів;
– присутня складність первинного налаштування для юзерів-новачків.

2 ОГЛЯД ОСОБЛИВОСТЕЙ ПОБУДОВАНОЇ СИСТЕМИ IOT

Для забезпечення мінімального набору функцій керування освітленням у системі IOT, а саме вимикання світла за допомогою вимикача або віддалено через смартфон, потрібно вибрати правильний стек апаратного та програмного забезпечення.

2.1 Опис вибраної платформи NodeMCU

NodeMCU – це прошивка з відкритим вихідним кодом на основі Lua для WiFi SOC ESP32 та ESP8266 від компанії Espressif. Вона використовує файлову систему SPIFFS на основі вбудованої флешпам’яті. NodeMCU реалізовано на C, а версія ESP32 базується на Espressif ESP-IDF.

Спочатку прошивку було розроблено як супровідний проєкт до популярних модулів розробки NodeMCU на базі ESP8266, але зараз він підтримується спільнотою, і тепер прошивку можна запускати на будь-якому модулі ESP [4].

2.1.1 Огляд мікроконтролера ESP32

ESP32 – це система на чіпі, яка інтегрує в собі наступні функції:

- Wi-Fi (діапазон 2, 4 ГГц);
- Bluetooth (діапазон частот до 2,5 ГГц);
- два високопродуктивних 32-бітних процесорних ядра Xtensa LX6;
- співпроцесор з наднизьким енергоспоживанням;
- кілька периферійних пристроїв, включаючи ADC, DAC, SPI, I2C, UART, PWM та сенсорні входи.

ESP32, виготовлений за технологією 40 нм, забезпечує надійну, високоінтегровану платформу, яка допомагає задовольнити постійні вимоги до ефективного використання енергії, компактного дизайну, безпеки, високої продуктивності та надійності.

Основні характеристики ESP32:

– частота процесора до 240 МГц;
– оперативна пам’ять 520 КБ SRAM;
– підтримка флешпам’яті до 16 Мб;
– підтримка зовнішньої RAM до 8 Мб;
– оперативна пам’ять 520 КБ SRAM;
– підтримка флешпам’яті: до 16 Мб;
– діапазон робочих температур: -40°C до +125°C;
– живлення: 2,2-3,6 В.

Мікросхема ESP32 має 48 виводів з різними функціями. Але не всі виводи доступні на всіх розробницьких платах ESP32, а деякі виводи не можуть бути використані [5].

На рисунку 2.1 показано розводку виводів ESP-WROOM-32.

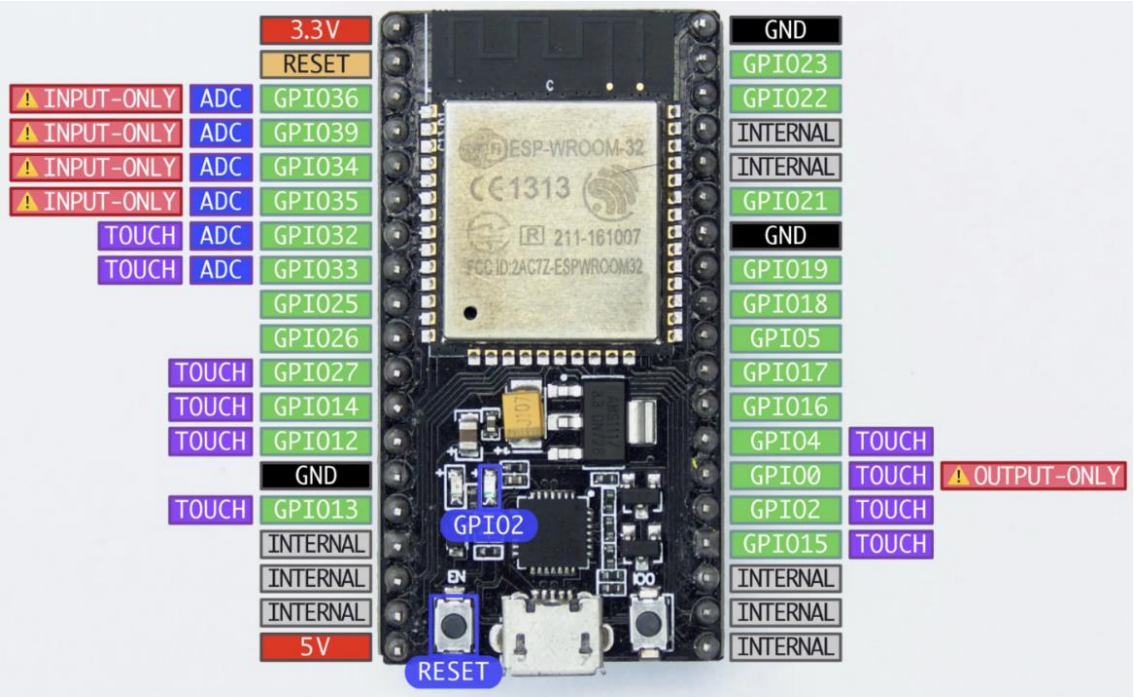

Рисунок 2.1 – Схема розташування виводів (Pinout) у NodeMCU-ESP32

2.1.2 Порівняльна характеристика мікроконтролерів ESP32 та ESP8266

ESP32 та ESP8266 – це дешеві Wi-Fi модулі, які ідеально підходять для проєктів «зроби сам» у сфері Інтернету речей і домашньої автоматизації.

Обидва чіпи мають 32-бітний процесор. ESP32 – це двоядерний процесор з тактовою частотою від 160 до 240 МГц, тоді як ESP8266 – одноядерний процесор з тактовою частотою 80 МГц (див. табл. 2.1).

Таблиця 2.1 – Порівняльна характеристика MCU ESP8266 та ESP32

Характеристика	ESP8266	ESP32
MCU	Xtensa Single-core 32-bit L106	Xtensa Dual-Core 32-bit LX6 with 600 DMIPS
802.11 b/g/n Wi-Fi	HT20	HT40
Bluetooth	Відсутній	Bluetooth 4.2 and BLE
Типова частота	80 МГц	160 МГц
SRAM	Відсутній	Присутній
Flash	Відсутній	Присутній
GPIO	17	34
Програмний PWM	8 каналів	16 каналів
SPI/I2C/I2S/UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	Відсутній	Присутній
Ethernet MAC	Відсутній	Присутній
Сенсорний датчик	Відсутній	Присутній
Датчик температури	Відсутній	Присутній
Датчик Холла	Відсутній	Присутній
Робоча температура	Від -40 ºС до 125 ºС	Від -40 ºС до 125 ºС

Обидва мікроконтролери є популярними між розробників в галузі Інтернет речей. Ці модулі виробляє компанія Espressif Systems, що робить їх дуже схожими один між одним. Однак ESP32 в певною мірою перевищує свого попередника за багатьма параметрами, що дозволяє використовувати його для більш комплексних проєктів.

ESP8266 став одним із перших Wi-Fi модулів, який був доступний за ціною для кожного. Він оснащений мікроконтролером Xtensa Single-core 32-bit L106 і працює на частоті 80 MHz. ESP8266 підтримує Wi-Fi 802.11 b/g/n і має 17 GPIO. Цей модуль також підтримує PWM на 8 каналах, а також інтерфейси SPI, I2C, I2S та UART.

Загалом ESP8266 є чудовим вибором для проєктів IoT, але ESP32 може бути використаний для більш гнучкої роботи та облегшити створення засобів для керування приладами через мережу інтернет.

2.2 Опис програмної частини мобільного додатку для керування освітленням

Програмна частина проєкту зосереджена навколо інтегрованого середовища розробки для платформи Android – Android Studio, мови програмування Kotlin та платформи Node.js. Крім того, в апаратній частині використано обчислювальну платформу Arduino з елементами мов C та C++.

2.2.1 Мова програмування Kotlin

Kotlin – це проста, кросплатформна мова програмування, яка працює на основі JVM (Java Virtual Machine). Ця мова підтримує як і об’єктноорієнтоване, так і функціональне програмування. Котлін має схожий синтаксис та концепції з іншими мовами, включаючи C#, Java та Scala, серед багатьох інших.

Kotlin управляється Kotlin Foundation, групою, створеною JetBrains і Google, завданням якої є просування і подальший розвиток мови. Kotlin офіційно підтримується Google для розробки Android, що означає, що документація та інструментарій для Android розроблені з урахуванням Kotlin [6].

2.2.2 Елементи Arduino в апаратній частині

Програмування апаратної частини проєкту полягає в прошивці мікроконтролера ESP-32 за допомогою Arduino IDE. Для повноцінного використання плати потрібно налаштувати декілька важливих компонентів.

Налаштування Wi-Fi. Модуль Wi-Fi вже інтегровано в мікроконтролер ESP-32, тому у програмі потрібно налаштувати цей модуль за допомогою задання SSID для підключення до мережі, SSID для точки доступу (AP) та паролі.

Для створення вебсервера використовується бібліотека «WebServer». Створення відбувається на стандартному порту для HTTP (див. рис. 2.2).

```
WebServer HTTP(80);
```

Рисунок 2.2 – Налаштування пінів ESP-32

Налаштування пінів. Для підключення реле до ESP-32 використовуються цифрові піни, до яких підключені електричні пристрої, такі як лампочка, світлодіод тощо. На рисунку 2.3 надано налаштування пінів у коді.

```
const int relayPins[] = {25, 26, 27, 14, 23, 22, 21, 19};
const int numRelays = sizeof(relayPins) / sizeof(relayPins[0]); /
int relayStates[] = {HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, HIGH};
```

Рисунок 2.3 – Налаштування пінів ESP-32

ESP-32 загалом має 48 доступних GPIO пінів, тому потрібно вибрати деякі з них, зважаючи на апаратну конфігурацію та дизайн плати. Масив relayStates встановлює початковий стан кожного з пінів як «HIGH», що, у цьому випадку, означає вимкнений.

Основні функції. Функція «setup» виконує основні налаштування, які необхідні для запуску системи (див. рис. 2.4).

```
 void setup {
dht.begin();
  pinMode(Light_Pin, INPUT);
  loadArraysFromEEPROM();
  for (int i = 0; i < numRelays; i++) {  pinMode(Voltage_Sensor_Pin[i],INPUT); }
  for (int i = 0; i < numRelays; i++) {
  pinMode(relayPins[i], OUTPUT);
  digitalWrite(relayPins[i], relayStates[i]);
  } WIFIinit(); HTTP_init();
}
```

Рисунок 2.4 – Функція «setup»

Функція «loop» виконується безперервно, перевіряючи стан світла, і є основним циклом програми, що керує реле (див. рис. 2.5).

```
void loop() {
  HTTP.handleClient();
  if (digitalRead(Light_Pin) == 0) {
    for (int i = 0; i < numRelays; i++) {
      relayStates[i] = digitalRead(relayPins[i]);
    }
    for (int i = 0; i < numRelays; i++) {
      Voltage_Sensor[i] = digitalRead(Voltage_Sensor_Pin[i]);
    }
  } else {
    saveArraysToEEPROM();
  }
}
```

Рисунок 2.5 – Функція «loop»

2.3 Реалізація сервера на Node.js для тестування

Для тестування роботи додатка варто реалізувати тестовий сервер на базі Node.js. Такий сервер потрібен для симуляції роботи апаратної частини проєкту при відсутності безпосереднього доступу до неї.

Основна функціональна можливість цього сервера – обробка HTTP-запитів, які надсилає на нього додаток для отримання та налаштування даних з датчиків.

Загалом, сервер дозволяє виконувати такі дії:

– перевірка стану сервера;
– симуляція вимірювання температури та вологості;
– управління пінами реле;
– обробка некоректних запитів.

На рисунках 2.6-2.9 наведено приклади реалізації роботи сервера.

Запит GET до кореневого маршруту повертає повідомлення, що сервер працює, що дозволяє перевірити його доступність (див. рис. 2.6).

```
app.get('/', (req, res) => {
  res.send('Server is running.');
});
```

Рисунок 2.6 – Код надсилання сервером запита GET

POST-запит до маршруту «/t» повертає згенеровані випадковим чином значення температури та вологості, а також стан реле, відповідні показники сенсорів напруги (див. рис. 2.7).

Сервер обробляє POST-запити для включення та виключення реле за допомогою маршрутів у форматі «/<pin>on» і «/<pin>off». При цьому в логах сервера відображається інформація про виконані дії (див. рис. 2.8).

Усі некоректні POST-запити обробляються на маршруті «*», повертаючи статус 404 і повідомлення про відсутність такого кінцевого пункту (див. рис. 2.9).

```
app.post('/t', (req, res) => {
  baza(res);
});

function baza(res) {
  let temperature = getRandomInt(-25, 25);
  let humidity = getRandomInt(0, 100);
  const webString = relayStates
    .map((state, index) => `${index},${state},${Voltage_Sensor[index]}`)
    .join(',');
  console.log(webString);
  res.status(200).send(`${temperature},${humidity},${webString}`);
}
```

Рисунок 2.7 – Код з симуляцією даних та POST-запит з їх надсиланням

```
relayPins.forEach(pin => {
  app.post(`/${pin}on`, (req, res) => {
    res.sendStatus(200);
    console.log(pin + "on");
  });

  app.post(`/${pin}off`, (req, res) => {
    res.sendStatus(200);
    console.log(pin + "off");
  });
});
```

Рисунок 2.8 – Код надсилання сервером запита GET

```
app.post('*', (req, res) => {
    console.log("someone trying to get to wrong post endp");
res.status(404).send('Endpoint not found');
});
```

Рисунок 2.9 – Код надсилання сервером запита GET

Для взаємодії з цим сервером розробнику потрібно лише запустити його на локальному комп’ютері або в хмарному середовищі та налаштувати додаток на взаємодію з ним. Саме такий підхід дозволяє придати розробнику чималої ефективності при своїй роботі.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ КЕРУВАННЯ ПРИСТРОЯМИ ІНТЕРНЕТУ РЕЧЕЙ

Мобільний додаток реалізовано мовою програмування Kotlin в інтегрованому середовищі розробки Android Studio під мобільну операційну систему Android. Крім того, задля тестування мобільного додатка було створено тестовий сервер на основі платформи Node.js.

Схема всього проєкту, який був реалізований, представлена на рисунку 3.1.

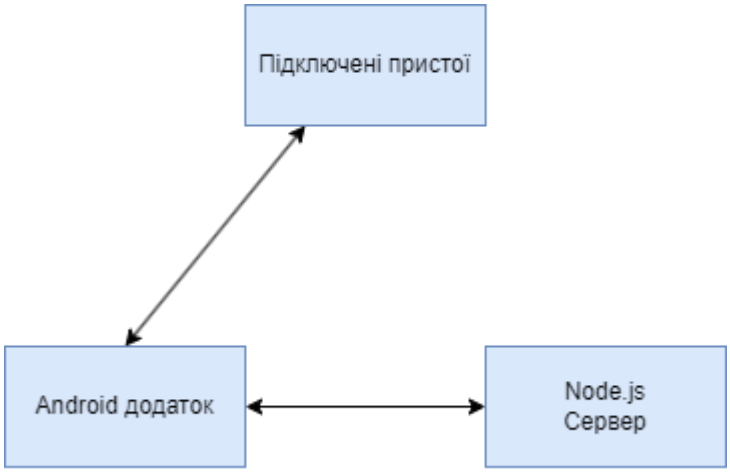

Рисунок 3.1 – Загальна схема роботи проєкту

Отже, на рисунку 3.1 можна побачити двосторонній зв’язок між Android додатком та апаратними приладами й Node.js сервером.

3.1 Реалізація мобільного додатку

Розробка мобільних додатків – це комплексне дослідження взаємодії інтерфейсу користувача та програмної реалізації, що потребує інтегрованого підходу.

3.1.1 Архітектура додатка

Архітектура додатка базована на принципах розділення відповідальності, простоти використання та зручності управління. Основні компоненти додатка містять наступні модулі.

Головна активність (MainActivity). Клас Activity є ключовим компонентом програми для Android, а спосіб запуску та компонування активностей є фундаментальною частиною прикладної моделі платформи. На відміну від парадигм програмування, в яких додатки запускаються за допомогою методу main(), система Android ініціює код в екземплярі Activity шляхом виклику спеціальних методів зворотного виклику, які відповідають певним етапам його життєвого циклу [7].

«MainActivity» є основною активністю мобільного додатку. Її головні обов’язки включають:

– ініціалізація основних компонентів (див. рис. 3.2);
– обробка натискань кнопок для керування реле;
– регулярне оновлення даних про температуру та вологість через використання coroutine.

```
private lateinit var binding: ActivityMainBinding
private val networkHelper = NetworkHelper()
private lateinit var dataStorageHelper: DataStorageHelper
private lateinit var chartHelper: ChartHelper
```

Рисунок 3.2 – Фрагмент коду з ініціалізацією компонентів

NetworkHelper. NetworkHelper відповідає за мережеву взаємодію додатку. Цей клас використовує бібліотеку OkHttpClient для здійснення HTTP-запитів. Основні методи NetworkHelper включають post (див. рис. 3.3) та getData (див. рис. 3.4). Метод getData виконує POST-запит для отримання даних про температуру та вологість з сервера. Він надсилає запит на заданий

URL і повертає відповідь у вигляді рядка, якщо запит був успішним. Метод post здійснює асинхронні POST-запити для керування реле, надсилаючи відповідні команди на сервер. Для цього використовується coroutine, яка запускає запит в окремому потоці, що дозволяє уникнути блокування головного потоку додатка. Всі результати запитів обробляються з використанням обробки винятків.

```
fun post(ip: String?, port: String, post: String) {
    CoroutineScope(Dispatchers.IO).launch {
        try {
            val success = sendPostRequest("http://$ip:$port/$post")
            if (!success) {
                Log.d("postError", " Помилка виконання запиту")
            }
        } catch (e: Exception) {
            Log.d("postError", " Помилка виконання запиту: ${e.message}")
        }
    }
}
```

Рисунок 3.3 – Метод post в класі NetworkHelper

```
fun getData(ip: String?, port: String): String? {
    val requestBody: RequestBody = "".toRequestBody("text/plain".toMediaType())
    val request = Request.Builder()
        .url("http://$ip:$port/t")
        .post(requestBody)
        .build()
    client.newCall(request).execute().use { response ->
        return if (response.isSuccessful) {
            response.body?.string()
        } else {
            null
        }
    }
}
```

Рисунок 3.4 – Метод getData в класі NetworkHelper

DataStorageHelper. Цей модуль забезпечує збереження та доступ до даних у сховищі. Основні функції включають:

– збереження та отримання IP-адреси сервера з поля для введення;
– збереження та отримання даних про температуру та вологість з апаратної частини проєкту (див. рис. 3.5);
– конвертація даних у масиви для подальшої обробки.

```
private fun saveData(key: String, value: String) {
    val currentData = pref.getString(key, "") ?: ""
    val dataList = currentData.split(" ").toMutableList()

    while (dataList.size >= 10) {
        dataList.removeAt(0)
    }

    dataList.add(value)
    val newData = dataList.joinToString(" ")

    pref.edit().putString(key, newData).apply()
}
```

Рисунок 3.5 – Метод saveData в класі DataStorageHelper

Метод saveData відповідає за збереження даних у SharedPreferences з обмеженням на кількість збережених записів. Вона приймає два параметри: key, який визначає ключ для збереження даних, та value, яке є новим значенням для збереження. Спочатку метод зчитує поточні дані, збережені під цим ключем, у вигляді рядка. Далі цей рядок розбивається на список окремих значень і додається до кінця списку. Такий формат роботи метода потрібен для подальшого її використання у виводі в графіку. Всі зміни застосовуються негайно завдяки методу apply, що дозволяє асинхронно зберегти дані без блокування основного потоку.

ChartHelper. ChartHelper відповідає за відображення графіка температури та вологості з використанням бібліотеки MPAndroidChart.

Основні функції включають:

– ініціалізація графіка та налаштування даних для відображення;
– оновлення графіка новими даними;
– налаштування осей графіка для зручного відображення (див. рис. 3.6).

```
fun setLineChartData() {
   val temperatureArray = dataStorageHelper.getTemperatureArray()
   val humidityArray = dataStorageHelper.getHumidityArray()
   val entries1 = temperatureArray.mapIndexed { index, value -> Entry(index.toFloat(), value.toFloat()) }
   val entries2 = humidityArray.mapIndexed { index, value -> Entry(index.toFloat(), value.toFloat()) }
   val lineDataSet1 = LineDataSet(entries1, "Temperature").apply {
      color = Color.RED
      valueTextColor = Color.BLACK
   }
   val lineDataSet2 = LineDataSet(entries2, "Humidity").apply {
      color = Color.BLUE
      valueTextColor = Color.BLACK
   }
   lineChart.data = LineData(lineDataSet1, lineDataSet2)
   lineChart.setTouchEnabled(true)
   lineChart.setPinchZoom(true)
   configureAxis()
   lineChart.invalidate()
}}
```

Рисунок 3.6 – Метод setLineChartData в класі DataStorageHelper

Метод setLineChartData в класі ChartHelper відповідає за налаштування та відображення даних на лінійному графіку. Спочатку він отримує масиви температури та вологості з DataStorageHelper, перетворюючи їх у списки об’єктів Entry, які представляють точки на графіку. Далі створюються два набори даних (LineDataSet), один для температури з червоним кольором, інший для вологості з синім кольором, кожен з яких налаштовується для відображення значень на графіку. Ці набори даних додаються до об’єкта

LineData, який потім призначається графіку. Графік налаштовується для підтримки жестів масштабування і прокручування, після чого викликається метод configureAxis для налаштування осей: вісь X розташовується внизу, а права вісь Y вимикається. Нарешті, викликається метод invalidate, щоб оновити графік та відобразити нові дані.

3.1.2 Інтерфейс користувача

UI-дизайн – процес візуалізації прототипу, який розробили на основі досвіду користувача і дослідження цільової аудиторії. Він містить в собі роботу над графічною частиною інтерфейсу: анімацією, ілюстраціями, кнопками, меню, слайдерами, фотографіями та шрифтами [8].

У випадку цього Android додатка, UI визначений за допомогою XML розмітки, яка описує розміщення та вигляд кожного елемента на екрані.

На рисунку 3.7 можна побачити кореневий елемент ConstraintLayout, який визначає загальну структуру UI. ConstraintLayout дозволяє створювати гнучкі та адаптивні розмітки, що підлаштовуються під різні розміри екранів.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white">
```

Рисунок 3.7 – Макет з обмеженнями для Android додатка

Далі представлено декілька елементів, таких як EditText та Button, які дозволяють користувачу вводити дані та взаємодіяти з додатком.

На рисунку 3.8, EditText використовується для введення IP-адреси. Цей елемент має певні параметри, такі як ems, hint (підказка для користувача) та інші, що налаштовують його зовнішній вигляд та поведінку.

```
<EditText
   android:id="@+id/edIp"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:layout_marginStart="8dp"
   android:layout_marginEnd="8dp"
   android:ems="10"
   android:hint="ip address"
   android:inputType="text"
   android:padding="15dp"
   app:layout_constraintBottom_toTopOf="@+id/bSave"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintHorizontal_bias="1.0"
   app:layout_constraintStart_toStartOf="parent" />
```

Рисунок 3.8 – XML-розмітка для текстового поля

На рисунку 3.9 можна побачити Button для збереження введеної IP-адреси. Він має текст “Save ip” та обмеження, щоб правильно розміститися на екрані відносно EditText.

```
<Button
   android:id="@+id/bSave"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:layout_marginBottom="16dp"
   android:text="Save ip"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintEnd_toEndOf="@+id/edIp"
   app:layout_constraintHorizontal_bias="0.0"
   app:layout_constraintStart_toStartOf="@+id/edIp" />
```

Рисунок 3.9 – XML-розмітка для текстового поля

Крім того, інтерфейс включає елементи, такі як LinearLayout, TextView та навіть графік LineChart з бібліотеки MPAndroidChart. Ці елементи разом створюють повний інтерфейс користувача, з яким користувач може взаємодіяти.

Отож, після виконання всіх дій для забезпечення сучасного UI, макет інтерфейс має такий вигляд (див. рис. 3.10).

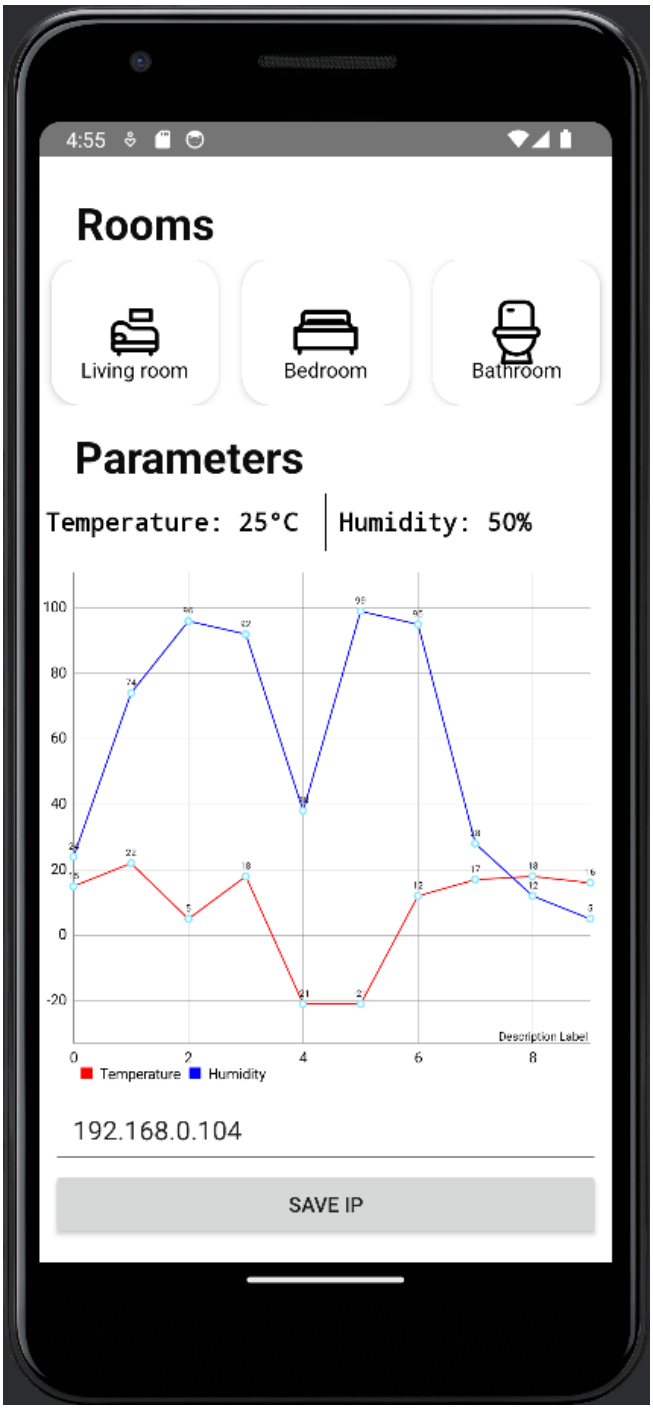

Рисунок 3.10 – Загальний вигляд інтерфейсу користувача

3.1.3 Взаємодія з пристроями Інтернету речей

Взаємодія з пристроями Інтернету речей відбувається завдяки POST-запитам на сервер апаратної частини. Сервер, своєю чергою, відправляє дані про температуру, вологість та стан реле, які отримуються у вигляди строки.

Надсилання POST-запитів для керування пристроями:

– кожен пристрій (реле) керується окремим POST-запитом;
– запит містить IP-адресу сервера, порт і команду для керування станом реле (увімкнути, вимкнути, отримати стан);
– при натисканні кнопки, метод відправляє POST-запит з відповідною командою (див. рис. 3.11).

```
private fun onClickListener(pin: String, index: Int): View.OnClickListener {
    return View.OnClickListener { view ->
        toggleButtonBackground(view)
        val state = if (buttonStates[index]) "off" else "on"
        networkHelper.post(dataStorageHelper.getIp(), defaultPort, "$pin$state")
        buttonStates[index] = !buttonStates[index]
    }
```

Рисунок 3.11 – Метод onClickListener

У цьому прикладі, метод onClickListener приймає пін реле та індекс, створює обробник натискання, який змінює стан реле та відправляє POST-запит.

Отримання даних з сервера.

– дані про температуру, вологість та стан реле запитуються у серверу кожні 5 секунд за допомогою GET-запиту;
– відповідь сервера обробляється для видалення зайвих символів, розділення на окремі значення та збереження їх у SharedPreferences;
– отримані значення оновлюють інтерфейс користувача та графік (див. рис. 3.12).

```
private fun getTemperatureAndHumidity() {
    CoroutineScope(Dispatchers.IO).launch {
        try {
            val data = networkHelper.getData(dataStorageHelper.getIp(), defaultPort)
            if (data != null) {
                val cleanedData = data.replace("\"", "")
                val (temp, humidity, a, b, c) = cleanedData.split(",").map { it.toInt() }
                dataStorageHelper.saveTemperature(temp.toString())
                dataStorageHelper.saveHumidity(humidity.toString())

                runOnUiThread {
                    chartHelper.updateLineChartData()
                    showTemperatureAndHumidity(temp, humidity)
                }
            } else {
                Log.d("server-error", "Получен пустой ответ от сервера")
            }
        } catch (e: Exception) {
            Log.d("server-error", "Ошибка получения данных о температуре и влажности", e)
        }
    }
}
```

Рисунок 3.12 – Метод getTemperatureAndHumidity

Тут, метод getTemperatureAndHumidity здійснює GET-запит до сервера для отримання даних про температуру і вологість. Дані обробляються та відображаються в інтерфейсі користувача.

Обробка даних та оновлення інтерфейсу:

– отримані дані про температуру та вологість зберігаються у локальному сховищі для подальшого використання;

– дані відображаються у текстових полях та оновлюють графік, що дає можливість користувачу спостерігати за змінами в реальному часі (див. рис. 3.13);

```
private fun showTemperatureAndHumidity(temp: Int, humidity: Int) {
    binding.tvTemperature.text = "Temperature: $temp Cº"
    binding.tvHumidity.text = "Humidity: $humidity %"
}
```

Рисунок 3.13 – Метод showTemperatureAndHumidity

3.1.4 Управління сесіями

Управління сесіями в додатку здійснюється за допомогою використання SharedPreferences. Клас сховища дозволяє зберігати невеликі обсяги даних у вигляді ключ-значення, що робить його ідеальним для зберігання таких даних, як IP-адреса сервера, температура, вологість та інших параметрів.

SharedPreferences використовується у модулях NetworkHelper та DataStorage, що відповідає нормам розділення відповідальності.

Збережені дані використовуються у додатку для встановлення налаштувань та взаємодії з пристроями IoT. Наприклад, IP-адреса сервера використовується для надсилання на нього запитів (див. рис. 3.14).

```
networkHelper.post(dataStorageHelper.getIp(), defaultPort, "$pin$state")
```

Рисунок 3.14 – Приклад роботи з SharedPreferences

3.2 Тестування мобільного додатку

Тестування мобільного додатку є важливим етапом розробки, який дозволяє виявити помилки та перевірити коректність взаємодії додатка з сервером та пристроями Інтернету речей (IoT). Для цього тестування було використано сервер, реалізований на Node.js з використанням Express, який симулює роботу апаратної частини.

Таблиця 3.2 містить основні кроки та очікувані результати для тестування функціональності мобільного додатку, зокрема керування реле та отримання даних про температуру і вологість. Цей тест-кейс допоможе переконатися у коректній взаємодії додатка з сервером та, надалі, з пристроями Інтернету речей.

Таблиця 3.2 – Тест-кейс «Керування реле та отримання даних про температуру і вологість»

Номер дії	Дія	Очікуваний результат
1	Запустити сервер Node.js, що відповідає за симуляцію роботи апаратної частини.	Сервер запущений та працює на порту 3000. В консолі відображається повідомлення «Server is running on port 3000».
2	Встановити, або налаштувати емулятор та запустити мобільний додаток на Android пристрої.	Мобільний додаток запущено, відображається головний екран з кнопками для керування реле та полями для введення IP-адреси та відображення температури та вологості.
3	Ввести IP-адресу сервера в поле для введення IP-адреси та натиснути кнопку «Зберегти».	IP-адреса збережена, відображається підтвердження.
4	Натиснути на кнопку “bLed1” для увімкнення першого реле (пін 25).	Відправляється POST-запит на сервер за адресою /25on. В консолі сервера відображається повідомлення «25on».
5	Натиснути на кнопку “bLed1” для вимкнення першого реле (пін 25).	Відправляється POST-запит на сервер за адресою /25off. В консолі сервера відображається повідомлення “25off”.
6	Зачекати 5 секунд для автоматичного отримання даних про температуру та вологість.	Відправляється GET-запит на сервер за адресою /t. Сервер генерує випадкові значення температури та вологості й повертає їх у відповідь. Відображаються отримані дані.

Продовження таблиці 3.2

Номер дії	Дія	Очікуваний результат
7	Перевірити відображення даних про температуру та вологість у мобільному додатку.	У полі температури відображається “Temperature: <випадкове значення> Cº”, у полі вологості відображається “Humidity: <випадкове значення> %”.
8	Повторити кроки 4-9 для всіх реле (піни 25, 26, 27) та перевірити коректність їх увімкнення/вимкнення.	Всі реле успішно увімкнені та вимкнені відповідно до натискання кнопок, відповідні повідомлення відображаються в консолі сервера (див. рис. 3.15).
9	Перевірити стабільність роботи мобільного додатку при тривалому використанні (напр., протягом 30 хвилин).	Мобільний додаток продовжує працювати, дані оновлюються кожні 5 секунд, керування реле працює коректно.

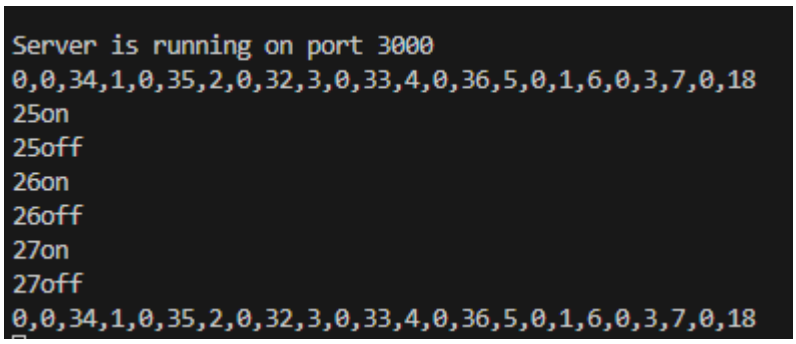

Рисунок 3.15 – Результат тест-кейсу у консолі сервера для тестування

ВИСНОВКИ

У кваліфікаційній роботі «Розробка мобільного додатку для керування пристроями інтернету речей» було досягнуто основної мети – створено мобільний додаток для керування освітленням на базі мікроконтролера ESP32 та платформи NodeMCU. У процесі роботи було виконано аналіз наявних рішень на ринку мобільних додатків для віддаленого керування освітленням та приладами Інтернету речей загалом, що дозволило визначити основні вимоги та критерії для розробки власного додатка.

Основні результати дослідження та розробки включають:

- проведено порівняльний аналіз сучасних мобільних додатків для віддаленого керування освітленням, що дозволило визначити найкращі практики для реалізації даного проєкту;
- виконано огляд вибраної платформи NodeMCU та мікроконтролера ESP32, включаючи порівняльну характеристику з іншими популярними мікроконтролером ESP8266;
- розроблено програмну частину мобільного додатку для керування освітленням, використовуючи мову програмування Kotlin;
- інтегровано елементи Arduino в апаратну частину системи;
- реалізовано сервер на Node.js для тестування додатка;
- створено інтуїтивно зрозумілий інтерфейс користувача для мобільного додатка;
- проведено тестування мобільного додатку.

Результати цієї роботи демонструють можливість успішної розробки та впровадження мобільного додатку для керування освітленням на базі сучасних технологій інтернету речей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Philips lighting. URL: https://www.lighting.philips.ua/consumer (дата звернення: 21.03.2024).
2. Lifx Integrations Directory. URL: https://www.lifx.com/blogs/integrations-directory (дата звернення: 21.03.2024).
3. Mi Home – Apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.xiaomi.smarthome&hl=uk&gl=US (дата звернення: 23.03.2024).
4. NodeMCU Documentation. URL: https://nodemcu.readthedocs.io/en/dev-esp32/ (дата звернення: 01.04.2024).
5. Espressif ESP32 Hardware Design Guidelines. URL: https://docs.espressif.com/projects/esp-hardware-design-guidelines/en/latest/esp32/product-overview.html (дата звернення: 08.04.2024).
6. Kotlin Overview. URL: https://developer.android.com/kotlin/overview (дата звернення: 19.04.2024).
7. Introduction to Activities. URL: https://developer.android.com/guide/components/activities/intro-activities (дата звернення: 10.05.2024).
8. Що таке UX/UI дизайн. URL: https://redstone.media/shcho-take-ux-ui-dysayn (дата звернення: 14.05.2024).

ДОДАТОК А

Реалізація мобільного додатка для керування освітленням

```
package com.example.myapplication

import android.content.SharedPreferences
import android.graphics.Color
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.myapplication.databinding.ActivityMainBinding
import com.example.myapplication.helpers.NetworkHelper
import com.example.myapplication.helpers.DataStorageHelper
import com.example.myapplication.helpers.ChartHelper
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.isActive
import kotlinx.coroutines.launch


class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private lateinit var pref: SharedPreferences
    private val networkHelper = NetworkHelper()
    private lateinit var dataStorageHelper: DataStorageHelper
```

```
private lateinit var chartHelper: ChartHelper
private val buttonStates = mutableListOf(false, false, false)
private val relayPins = listOf("25", "26", "27")
private val defaultPort = "3000"
private var job: Job? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
    pref = getSharedPreferences("MyPref", MODE_PRIVATE)
    dataStorageHelper = DataStorageHelper(pref)
    chartHelper = ChartHelper(this, dataStorageHelper)

    onClickSaveIp()
    binding.edIp.setText(dataStorageHelper.getIp())

    binding.apply {
        bLed1.setOnClickListener(onClickListener(relayPins[0], 0))
        bLed2.setOnClickListener(onClickListener(relayPins[1], 1))
        bLed3.setOnClickListener(onClickListener(relayPins[2], 2))
    }
    chartHelper.setLineChartData()

    job = CoroutineScope(Dispatchers.IO).launch {
        while (isActive) {
            getTemperatureAndHumidity()
            delay(5000)
        }
    }
```

```
}

override fun onDestroy() {
    super.onDestroy()
    job?.cancel()
}

private fun onClickListener(pin: String, index: Int): View.OnClickListener {
    return View.OnClickListener { view ->
        toggleButtonBackground(view)
        val state = if (buttonStates[index]) "off" else "on"
        networkHelper.post(dataStorageHelper.getIp(), defaultPort, "$pin$state")
        buttonStates[index] = !buttonStates[index]
    }
}

private fun toggleButtonBackground(button: View) {
    button.isSelected = !button.isSelected
    button.setBackgroundResource(
        if (button.isSelected) R.drawable.button_background_blue
        else R.drawable.button_background
    )
}

private fun onClickSaveIp() {
    binding.bSave.setOnClickListener {
        val ip = binding.edIp.text.toString()
        if (ip.isNotEmpty()) dataStorageHelper.saveIp(ip)
    }
}
```

```
    private fun getTemperatureAndHumidity() {
        Log.d("ip", dataStorageHelper.getIp().toString())
        CoroutineScope(Dispatchers.IO).launch {
            try {
                val data = networkHelper.getData(dataStorageHelper.getIp(),
defaultPort)
                if (data != null) {
                    val cleanedData = data.replace("\"", "")
                    val (temp, humidity, a, b, c) = cleanedData.split(",").map { it.toInt() }
                    dataStorageHelper.saveTemperature(temp.toString())
                    dataStorageHelper.saveHumidity(humidity.toString())

                    runOnUiThread {
                        chartHelper.updateLineChartData()
                        showTemperatureAndHumidity(temp, humidity)
                    }
                } else {
                    Log.d("server-error", "Получен пустой ответ от сервера")
                }
            } catch (e: Exception) {
                Log.d("server-error", "Ошибка получения данных о температуре и
влажности", e)
            }
        }
    }
    private fun showTemperatureAndHumidity(temp: Int, humidity: Int) {
        binding.tvTemperature.text = "Temperature: $temp Cº"
        binding.tvHumidity.text = "Humidity: $humidity %"
    }
}
```

```
class NetworkHelper {
    private val client = OkHttpClient()

fun getData(ip: String?, port: String): String? {
    val requestBody: RequestBody = "".toRequestBody("text/plain".toMediaType())
    val request = Request.Builder()
        .url("http://$ip:$port/t")
        .post(requestBody)
        .build()

    client.newCall(request).execute().use { response ->
        return if (response.isSuccessful) {
            response.body?.string()
        } else {
            null
        }
    }
}
    fun sendPostRequest(url: String): Boolean {
        val requestBody = "".toRequestBody("text/plain".toMediaType())
        val request = Request.Builder()
            .url(url)
            .post(requestBody)
            .build()

        client.newCall(request).execute().use { response ->
            return response.isSuccessful
        }
    }
```

```
    fun post(ip: String?, port: String, post: String) {
        CoroutineScope(Dispatchers.IO).launch {
            try {
                val success = sendPostRequest("http://$ip:$port/$post")
                if (!success) {
                    Log.d("postError", "Ошибка выполнения запроса")
                }
            } catch (e: Exception) {
                Log.d("postError", "Ошибка выполнения запроса: ${e.message}")
            }
        }
    }
}

class DataStorageHelper(private val pref: SharedPreferences) {
    fun getIp(): String? {
        return pref.getString("ip", "")
    }

    fun saveIp(ip: String) {
        pref.edit().putString("ip", ip).apply()
    }

    fun saveTemperature(temperature: String) {
        saveData("temperature", temperature)
    }

    fun saveHumidity(humidity: String) {
        saveData("humidity", humidity)
    }
```

```
    private fun saveData(key: String, value: String) {
        val currentData = pref.getString(key, "") ?: ""
        val dataList = currentData.split(" ").toMutableList()

        while (dataList.size >= 10) {
            dataList.removeAt(0)
        }

        dataList.add(value)
        val newData = dataList.joinToString(" ")

        pref.edit().putString(key, newData).apply()
    }

    fun getTemperatureArray(): IntArray {
        return getDataArray("temperature")
    }

    fun getHumidityArray(): IntArray {
        return getDataArray("humidity")
    }

    private fun getDataArray(key: String): IntArray {
        val dataString = pref.getString(key, "") ?: ""
        return dataString.split(" ").mapNotNull { it.toIntOrNull() }.toIntArray()
    }
}
```