Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: *Комп'ютерна система підсвітки сходів з віддаленим моніторингом*

Виконав: студент *IV* курсу, групи *CI-42*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

		Радій Р.І.
	(підпис)	(прізвище та ініціали)
Керівник		*Яцишин В.В.*
	(підпис)	(прізвище та ініціали)
Нормоконтроль		*Тиш Є.В.*
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		*Осухівська Г.М.*
	(підпис)	(прізвище та ініціали)
Рецензент		*Готович В.А.*
	(підпис)	(прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
« ___ » _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Радію Роману Івановичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Комп'ютерна система підсвітки сходів з віддаленим моніторингом*

Керівник роботи *Яцишин Василь Володимирович, к.т.н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*27*» *січня* 2025 року № *4.7-53*

2. Термін подання студентом завершеної роботи *18.06.2025 р.*

3. Вихідні дані до роботи *Мікроконтролер на базі ESP32, світлодіодна стрічка, Telegram-бот, хмарні сервіси AWS*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз практичних рішень та вимог технічного завдання щодо системи віддаленого керування підсвіткою сходів. 2. Проєктування комп'ютерної системи підсвітки сходів з віддаленим моніторингом 3. Програмна реалізація віддаленого моніторингу комп'ютерної системи підсвітки сходів. 4. Безпека життєдіяльності, основи охорони праці. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

1. *Існуючі рішення організації підсвітки сходів.*
2. *Структурна схема комп'ютерної системи.*
3. *Архітектура комп'ютерної системи.*
4. *Блок-схема алгоритму роботи комп'ютерної системи*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	*Пилипець М.І., д.т.н., проф., каф. МТ*		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*27.01 – 02.02*	*Викон.*
2.	*Аналіз практичних рішень та вимог технічного завдання щодо системи віддаленого керування підсвіткою сходів*	*02.02-03.03*	*Викон.*
3.	*Проєктування комп'ютерної системи підсвітки сходів з віддаленим моніторингом*	*04.03-25.04*	*Викон.*
4.	*Програмна реалізація віддаленого моніторингу комп'ютерної системи підсвітки сходів*	*25.04-18.05*	*Викон.*
5.	*Безпека життєдіяльності, основи охорони праці*	*19.05-28.05*	*Викон.*
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*28.05-07.06*	*Викон.*
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*9.06 – 15.06*	*Викон.*
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*16.06 – 22.06*	*Викон.*
9.	*Захист кваліфікаційної роботи бакалавра*	*25.06.2025*	*Викон.*

Студент _____ *Радій Роман Іванович*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Яцишин Василь Володимирович*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Радій Р.І. Комп’ютерна система підсвітки сходів з віддаленим моніторингом: робота на здобуття ступеня бакалавра: спец. 123 – комп’ютерна інженерія. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя.

Ключові слова: система, підсвітка, сходи, моніторинг.

У кваліфікаційній роботи спроєктовано та реалізовано комп’ютерну систему автоматичного керування підсвіткою сходів з підтримкою віддаленого моніторингу та налаштування.

Основним апаратним елементом системи є мікроконтролер ESP32, що забезпечує обробку даних з різних сенсорів — руху, присутності та освітленості, а також керування адресною світлодіодною стрічкою. За рахунок інтеграції з хмарними сервісами Amazon Web Services (AWS) та використання WebSocket-з’єднання, система дозволяє здійснювати віддалене керування освітленням за допомогою Telegram-бота, а також підтримує синхронізацію станів між пристроєм та сервером.

У межах реалізації було розроблено багатозадачне програмне забезпечення, яке включає обробку локального керування кнопками, збереження конфігурацій у файловій системі SPIFFS, обробку команд із хмари, зміни кольору підсвітки та перемикання між різними режимами візуалізації (статичне світло, ефекти "pong", "random", "checkers", "two colors" тощо).

ANNOTATION

Radii R.I. Computer-Based Stair Lighting System with Remote Monitoring: Bachelor's Graduation Thesis: speciality 123 — computer engineering. Ternopil: Ternopil Ivan Puluj National Technical University, 2025.

Keywords: system, lighting, stairs, monitoring.

In the qualification thesis, a computer-based system for automatic stair lighting control with support for remote monitoring and configuration was designed and implemented.

The core hardware component of the system is the ESP32 microcontroller, which handles data processing from various sensors — motion, presence, and ambient light — and controls an addressable LED strip. Through integration with Amazon Web Services (AWS) cloud services and the use of a WebSocket connection, the system enables remote lighting control via a Telegram bot and supports real-time state synchronization between the device and the server.

As part of the implementation, multitasking software was developed, including local button control handling, configuration storage using the SPIFFS file system, processing of cloud commands, dynamic color adjustment, and switching between different visualization modes (static light, "pong", "random", "checkers", "two colors" effects, etc.).

ЗМІСТ

					КС КРБ 123.233.00.00 ПЗ		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		*Радій Р.І.*			*Комп'ютерна система підсвітки сходів з віддаленим моніторингом*	*Літ.*	*Арк.* 6 / *Аркушів*
Перевір.		*Яцишин В.В.*					
Реценз.						*ТНТУ, каф. КС, гр. СІ-42*	
Н. Контр.		*Тиш Є.В.*					
Затверд.		*Осухівська Г.М.*					

ВСТУП

У сучасному світі концепція «розумного дому» стрімко трансформується із теоретичної моделі у практичну реальність. Інтелектуальні системи автоматизації дедалі частіше впроваджуються у повсякденне життя, покращуючи комфорт, безпеку та енергоефективність житлових і комерційних приміщень. Однією з таких систем є автоматизоване керування підсвіткою, зокрема, підсвіткою сходових маршів, яка не лише виконує функцію декоративного елементу, але й значно підвищує рівень безпеки у темний час доби.

Кваліфікаційна робота присвячена розробці комп'ютерної системи інтелектуального керування підсвіткою сходів із можливістю віддаленого моніторингу та керування. Основний фокус роботи спрямований на побудову енергоефективної та адаптивної системи освітлення, що реагує на наявність руху, рівень освітленості навколишнього середовища, а також дозволяє користувачеві змінювати режими підсвічування дистанційно через мобільний застосунок або Telegram-бот.

На основі мікроконтролера ESP32 система поєднує в собі функціональність локального управління через кнопки та дистанційного контролю за допомогою хмарних сервісів і бездротових технологій. Передбачена інтеграція з датчиками руху, присутності та освітленості дає змогу досягти адаптивної поведінки пристрою, що автоматично вмикає освітлення при необхідності та вимикає його, коли сходи не використовуються.

Актуальність теми зумовлена зростаючими вимогами до безпеки та комфорту житлових просторів, а також необхідністю енергоефективного управління освітленням. У межах даної роботи буде реалізовано повний цикл розробки системи починаючи від аналізу вимог і розробки апаратної платформи до написання програмного забезпечення та проведення тестування.

РОЗДІЛ 1 АНАЛІЗ ПРАКТИЧНИХ РІШЕНЬ ТА ВИМОГ ТЕХНІЧНОГО ЗАВДАННЯ ЩОДО СИСТЕМИ ВІДДАЛЕНОГО КЕРУВАННЯ ПІДСВІТКОЮ СХОДІВ

1.1 Аналіз вимог і технічного завдання до системи віддаленого керування підсвіткою сходів

Комп’ютерна система підсвітки сходів з віддаленим моніторингом повинна бути спроектована як інтерактивне комп’ютеризоване рішення, що забезпечує контроль і керування адресними світлодіодними стрічками. Систему пропонується реалізувати на базі мікроконтролера ESP32 з підтримкою Wi-Fi та Bluetooth, а також інтегрувати з популярним месенджером Telegram для реалізації інтуїтивно зрозумілого мобільного інтерфейсу.

Основною ціллю розробки є створення адаптивного освітлення, що реагує як на локальні дії користувача (через фізичні кнопки), так і на команди, надіслані з віддаленого пристрою. Ключовою особливістю є використання адресної світлодіодної стрічки WS2812B, кожним елементом якої можна програмно керувати. Це дозволяє реалізувати широкий спектр анімаційних ефектів, кольорових схем і режимів індикації.

На відміну від традиційних систем освітлення, проектована комп’ютерна повинна забезпечувати здатність адаптуватися до сценаріїв освітлення на основі команд користувача, зберігати конфігурацію у хмарному сховищі, обмінюватися інформацією з сервером у режимі реального часу за допомогою WebSocket-з’єднання.

Особливу увагу варто приділити доступності та надійності. Telegram-бот доступний у будь-який момент, а фізичні кнопки забезпечують дублювання

функціоналу у випадку втрати мережевого з’єднання. Пристрій повинен також підтримувати функцію автоматичного перепідключення до Wi-Fi та збереження останнього активного стану освітлення, що зробить його стабільним та зручним у повсякденному використанні.

Таким чином, система призначена не лише для реалізації візуального комфорту чи декоративного підсвічування, але й для демонстрації практичного застосування IoT-підходів у побутовому середовищі, об'єднуючи апаратне забезпечення, програмну логіку та хмарну інтеграцію в єдину функціональну платформу. Вона також є масштабованою, із можливістю керування кількома пристроями з одного інтерфейсу.

Основна мета створення системи моніторингу освітленням сходів полягає у підвищенні гнучкості, зручності та функціональності сучасних освітлювальних рішень шляхом поєднання локального та віддаленого керування, інтеграції з мобільним інтерфейсом (Telegram-ботом) та забезпечення двостороннього моніторингу через хмарну інфраструктуру.

Проєкт орієнтований на реалізацію системи, яка дозволяє користувачеві змінювати параметри підсвітки на базі WS2812B у режимі реального часу, створювати індивідуальні сценарії освітлення, а також отримувати актуальний стан пристрою незалежно від фізичної близькості до нього.

Досягнення поставленої мети можливе через виконання наступних задач:

– проведення аналітичного огляду наявних рішень керування освітленням та систем з Telegram-інтеграцією;

– розробка структурної та функціональної архітектури комп’ютеризованої системи керування адресною стрічкою на базі ESP32;

– обґрунтування вибору апаратного забезпечення (контролер, кнопки, стрічка WS2812B, джерело живлення);

– побудова схеми підключення апаратних компонентів з урахуванням енергетичних та комунікаційних характеристик;

– реалізація програмного забезпечення, яке забезпечує обробку команд з Telegram, взаємодію з WebSocket-сервером та керування світлодіодами;

– проведення тестування готової системи, вимірювання затримки реакції, перевірка стабільності зв’язку з мережею та оцінка надійності роботи у різних сценаріях.

Об’єктом дослідження у кваліфікаційній роботі є комп’ютерна система віддаленого моніторингу освітлення, що реалізується з використанням адресної світлодіодної стрічки WS2812B, мікроконтролера ESP32 та технологій інтернету речей (IoT). Система забезпечує як локальне, так і віддалене керування підсвіткою в режимі реального часу, що робить її універсальним рішенням для автоматизованого керування сценаріями освітлення.

Головною особливістю об'єкта є здатність виконувати інтерактивну зміну станів освітлення: колір, яскравість, режим за допомогою двох паралельних каналів управління, зокрема, фізичних кнопок, які підключені до контролера ESP32 і Telegram-бота, що працює через хмарну інфраструктуру (API Gateway + WebSocket + Lambda).

Система побудована на базі архітектури реального часу, яка використовує FreeRTOS для забезпечення паралельного виконання задач. Кожен світлодіод у стрічці програмується індивідуально, що дозволяє реалізовувати складні анімаційні ефекти та сценарії з високою точністю.

Особливу увагу в системі приділено адаптивності освітлення завдяки адресності WS2812B, дистанційному контролю через Telegram-інтерфейс та двосторонній синхронізації з сервером за допомогою WebSocket.

У ситуації втрати з’єднання з мережею або сервером, система автоматично переходить у локальний режим керування без втрати поточних налаштувань. Крім того, всі основні дії супроводжуються зворотним повідомленням у Telegram, що дозволяє користувачу мати повний контроль над станом пристрою.

Завдяки використанню модульного підходу, система може бути розширена шляхом підключення додаткових пристроїв або розширення сценаріїв, наприклад, додавання сенсорів освітленості для автоматичного вмикання підсвітки в темний час доби, або інтеграції голосових помічників (Google Assistant, Alexa).

Крім базових функцій, об'єкт дослідження також включає інтелектуальний моніторинг стану пристрою (через WebSocket-з’єднання з AWS), автоматичне відновлення Wi-Fi-з'єднання, систему захисту від несанкціонованого доступу через токени та шифрування даних.

Комп’ютерна система повинна проектуватися як багатокомпонентне IoT-рішення, яке реалізує сучасні підходи до інтеграції мікроконтролерів, хмарних технологій та зручних мобільних інтерфейсів управління у єдину платформу для використання в побутових, навчальних або комерційних середовищах.

Комп’ютерна система підсвітки сходів з віддаленим моніторингом – це багатофункціональна IoT-система, що забезпечує керування освітленням на основі подій, пов’язаних із присутністю користувача, а також дозволяє здійснювати моніторинг та конфігурацію у режимі реального часу. Загальні вимоги до системи поділяються на функціональні та нефункціональні, що охоплюють як апаратні, так і програмні аспекти її роботи.

До ключових функціональних можливостей системи належать:

– автоматичне вмикання та вимикання світлодіодної підсвітки при виявленні руху на сходах;

– поступове включення світла знизу вгору або згори вниз залежно від напрямку руху користувача;

– локальне керування режимами підсвітки за допомогою фізичних кнопок (наприклад, вибір сценарію освітлення);

– можливість дистанційного керування підсвіткою через мобільний застосунок або месенджер (наприклад, Telegram-бот);

– збереження поточного режиму освітлення та його автоматичне відновлення після перезавантаження;

– надсилання стану пристрою на сервер моніторингу через WebSocket у режимі реального часу;

– підтримка декількох сценаріїв підсвітки (анімовані, статичні, нічний режим);

– можливість ручного керування системою при відсутності з’єднання з мережею Wi-Fi.

Програмне забезпечення системи повинно забезпечувати:

– обробку сигналів з датчиків руху (HC-SR501) та освітленості з мінімальною затримкою (до 1 с);

– реалізацію анімаційного сценарію підсвітки з урахуванням напряму руху користувача;

– передачу інформації про стан системи (увімкнення, активний режим, рівень освітлення) через WebSocket до хмарної платформи (наприклад, AWS);

– реакцію на запити з Telegram-бота в межах до 3 секунд;

– логування подій, що включає фіксацію часу активації, активного режиму, статусу датчиків та користувацьких дій;

– підтримку багатозадачності за допомогою операційної системи реального часу (наприклад, FreeRTOS на ESP32).

До нефункціональних характеристик системи належать:

– зручність та ергономіка реалізуються через користувацький інтерфейс Telegram-бота, який повинен бути інтуїтивно зрозумілим;

– надійність – система має функціонувати без збоїв при постійному циклі увімкнення/вимкнення протягом не менше ніж 72 годин;

– доступність – забезпечення дублювання основних функцій шляхом локального керування, яке повинно працювати незалежно від доступу до Інтернету;

– захищеність – передбачає використання авторизаційних токенів при взаємодії з Telegram-ботом, шифрування даних між пристроєм та сервером;

– масштабованість – можливість додавання нових сценаріїв освітлення, підключення додаткових сенсорів або розширення стрічки;

– енергоефективність – зменшення яскравості в нічний час, оптимізація тривалості підсвітки.

1.2 Класифікація та загальна архітектура систем підсвітки сходів

Сучасні тенденції в автоматизації побутових та комерційних приміщень передбачають підвищення рівня комфорту, енергоефективності та безпеки. Одним з напрямів такої автоматизації є інтелектуальні системи освітлення, зокрема, системи підсвітки сходів з можливістю віддаленого керування та моніторингу. Ці рішення не лише покращують естетичне оформлення приміщень, а й виконують важливу функцію забезпечення видимості на сходах у нічний час або при поганому освітленні.

Дослідження показують, що використання адаптивних світлодіодних підсвіток дозволяє знизити споживання електроенергії на 30–50% у порівнянні з традиційними системами. Крім того, інтеграція віддаленого моніторингу забезпечує оперативний контроль стану системи та вчасне виявлення несправностей.

Системи підсвітки сходів можна класифікувати за такими ознаками:

- тип керування – може використовуватися локальне управління з використанням датчиків руху, кнопок, таймерів або віддалене – із застосуванням Wi-Fi, Bluetooth, Zigbee, GSM та інших безпровідних технологій передачі даних;
- тип освітлення – можна застосовуватися постійне або динамічне освітлення, що реагує на присутність рухомого об’єкта, час доби, рівень освітлення;
- рівень автоматизації – характеризується простими системи вмикання/вимикання та інтелектуальними системи, які реалізують алгоритмічне керування, або використовують елементи штучного інтелекту;
- зворотний зв'язок та моніторинг – існують системи з наявністю контролю стану освітлення та без нього.

У загальному випадку архітектуру сучасної системи підсвітки сходів можна представити у вигляді взаємодії таких компонентів:

- контролер управління, наприклад, Arduino, ESP32;

- датчики руху, освітленості, присутності;
- світлодіодні елементи, наприклад, світлодіодні стрічки, модулі;
- комунікаційні модулі, зокрема, Wi-Fi, GSM;
- хмарні платформи або мобільні додатки;
- модуль живлення.

У загальному випадку, архітектуру систем віддаленого керування підсвіткою сходів, можна представити у вигляді структурної схеми, як показано на рис. 1.1.

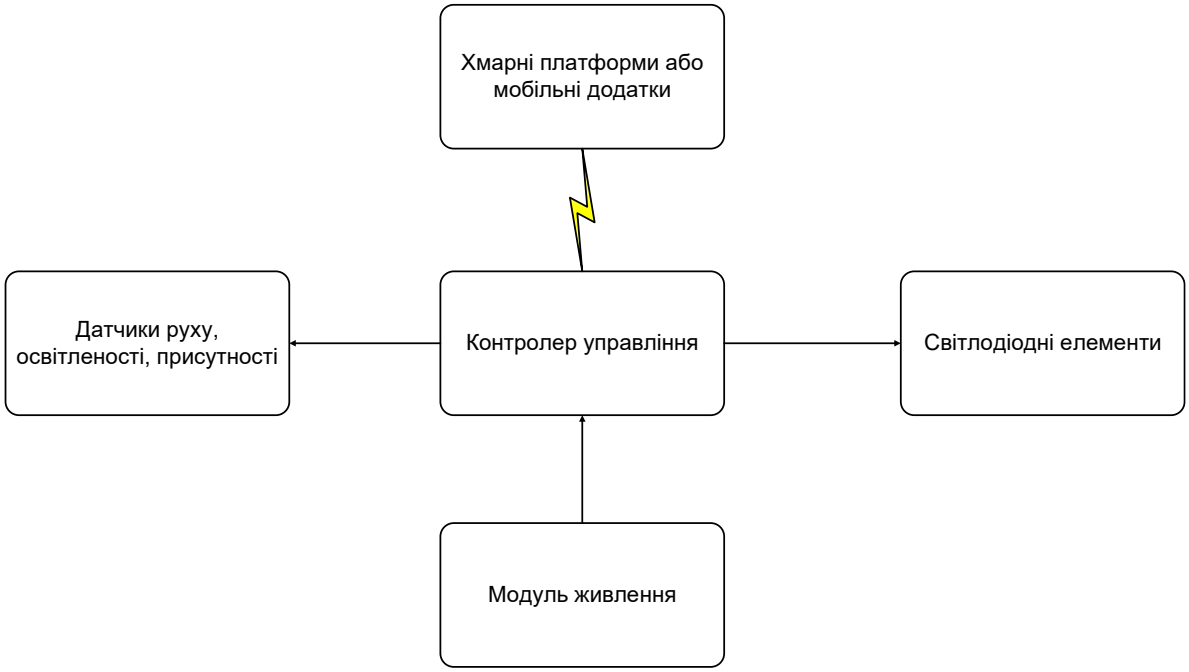

Рисунок 1.1 – Узагальнена структурна схема систем віддаленого керування підсвіткою сходів

Контролер управління є центральним пристроєм, що забезпечує керування сенсорами та обміном даним з мобільним додатком або хмарним сервісом. До датчиків руху належить PIR-сенсор. В якості датчиків освітленості можна використовувати фоторезистори, а присутності – інфрачервоні або ультразвукові сенсори. Світлодіодними елементами можуть виступати звичайні або адресні світлодіодні стрічки.

1.3 Огляд технологій віддаленого керування при керування підсвіткою сходів

Одним із найпоширеніших підходів є використання модулів ESP8266 або ESP32 з підтримкою Wi-Fi. Такі модулі працюють у поєднанні з протоколом MQTT для передавання даних до хмарного сервера. MQTT – легкий протокол публікації-підписки, який дозволяє ефективно передавати інформацію навіть за низької пропускної здатності.

В умовах, коли неможливо під’єднати пристрій до локальної мережі, застосовують GSM-модулі (SIM800L, A6, SIM900), які забезпечують передачу SMS або підключення до інтернету через мобільні мережі. Це забезпечує високу автономність, однак потребує SIM-карти та витрат на мобільний трафік.

Bluetooth та Zigbee технології використовуються переважно в локальних бездротових мережах. Наприклад, Zigbee є енергоефективним рішенням для багатозонних систем з декількома контролерами. Bluetooth частіше використовується для керування через мобільний додаток на короткій відстані.

Системи моніторингу підсвітки сходів дозволяють відображати стан кожного світлодіодного елементу, визначати несправності, зокрема, перегрів або втрату зв’язку, а також записувати дані про активність – час і тривалість увімкнення пдсвітки при проходженні по сходах

До найбільш популярних платформ для моніторингу за станом підсвітки сходів належать:

- Blynk – хмарна платформа для створення інтерфейсів керування IoT-пристроями з можливістю обміну даними через Wi-Fi або GSM;
- ThingSpeak – сервіс для обробки, зберігання та візуалізації телеметричних даних;
- Home Assistant – потужна платформа з відкритим кодом для створення повноцінної автоматизації "розумного дому".

Одним з варіантів для автоматичної підсвітки сходів є застосування мікроконтролера Arduino. Розробки на Arduino Uno або Nano з використанням

датчиків HC-SR501 (інфрачервоний рух) дозволяють створити просту та бюджетну систему з поступовим ввімкненням світлодіодної стрічки (рис. 1.2). Проте такі рішення не мають віддаленого доступу та не забезпечують моніторингу.

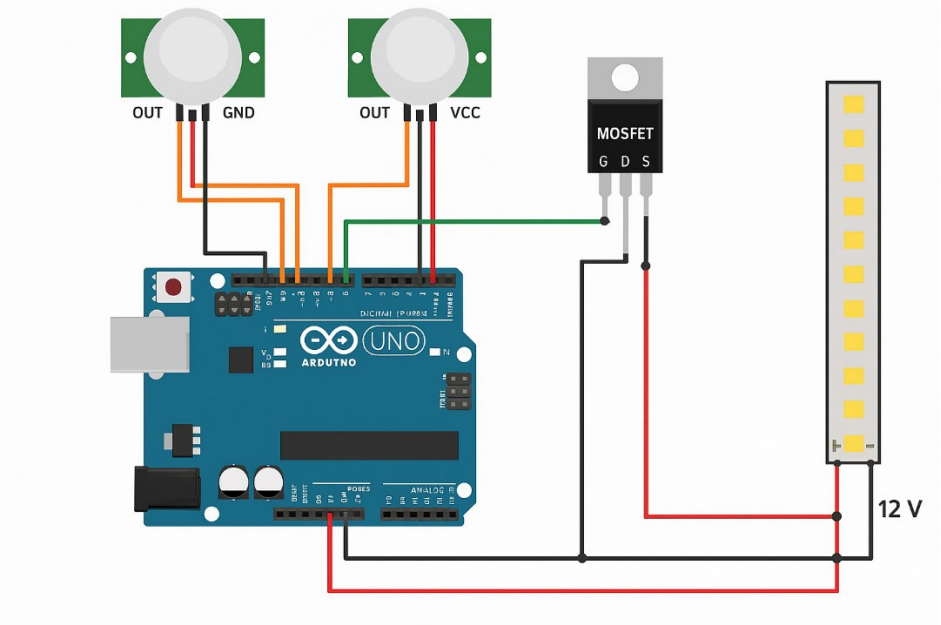

Рисунок 1.2 – Схема організації підсвітки сходів на базі Arduino

Принцип роботи такої системи передбачає наявність двох датчиків HC-SR501, які встановлюються на початку та в кінці сходового маршу. При спрацьовуванні будь-якого з двох сенсорів система визначає напрямок руху людини (вгору або вниз) і контролер управління Arduino поетапно вмикає сегменти світлодіодної стрічки, створюючи ефект "хвилі".

Після завершення проходу людини по сходах світло автоматично гасне в зворотному порядку. Затримка вимкнення може бути задана вручну, наприклад, 10 секунд неактивності.

Спрощений алгоритм керування підсвіткою сходів у цьому випадку передбачає виконання таких послідовних кроків, як постійне опитування стану обох HC-SR501. У випадку активації одного з датчиків виконується ідентифікація напрямку руху та активація функції поступового ввімкнення світлодіодів, наприклад, через програмну функцію analogWrite(), з підвищенням

яскравості. Після проходу, таймер відлічує 10–15 секунд і відбувається поступове вимкнення LED-стрічки. Система переходить у режим очікування нового спрацювання.

Серед переваг застосування такої системи варто виділити низьку вартість, простоту реалізації, забезпечення естетичного ефекту плавного освітлення. Обмеження, які накладаються на такі рішення, полягають у відсутності віддаленого моніторингу, обмеженій точності визначення напрямку, відсутності мобільного інтерфейсу та можливості зберігати лог подій.

Іншим прикладом реалізації системи віддаленого управління підствікою сходів є рішення на основі базі ESP32 та Blynk. ESP32 забезпечує підключення до Wi-Fi, а мобільний додаток Blynk дозволяє користувачу керувати підсвіткою, переглядати статус у реальному часі та отримувати сповіщення про аномалії. Це рішення є більш просунутим, однак потребує налаштування мережі, авторизації на сервері та постійного доступу до інтернету. На рис. 1.3 показано типову архітектуру системи організованої на основі ESP32 та Blynk.

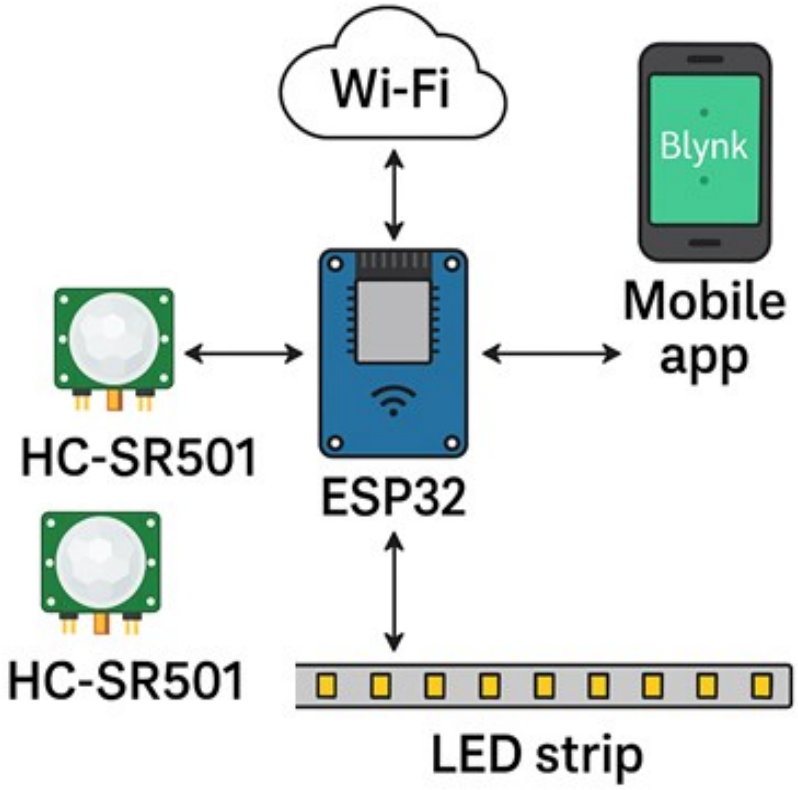

Рисунок 1.3 – Рішення щодо підсвітки на основі ESP32 та Blynk

Система, наведена на рис. 1.3, призначена для автоматичного ввімкнення світлодіодного підсвічування при виявленні руху на сходах, а також для віддаленого керування, моніторингу та отримання повідомлень через мобільний додаток.

Першим кроком алгоритму функціонування системи, представленої на рис. 1.3 є ініціалізація, що передбачає підключення ESP32 до локальної Wi-Fi мережі та виконання авторизації на сервері Blynk через конкретний токен. Далі встановлюються вхідні та вихідні піни: входи – два виводи для HC-SR501, вихід – ШІМ-ввиід для керування яскравістю світлодіодної стрічки.

Другий етап алгоритму виконує опитування датчиків, тобто постійно перевіряється стан кожного HC-SR501. Якщо активується датчик на вході (наприклад, нижній), то система починає плавне вмикання стрічки. Якщо активується верхній – аналогічна реакція для спуску.

За допомогою широтно-імпульсної модуляції забезпечується плавне ввімкнення світлодіодної стрічки, тобто стрічка поступово набирає яскравості і освітлення заданий час тримається увімкненим. Після тайм-ауту світло плавно гасне до нуля.

У мобільному застосунку можна увімкнути/вимкнути підствіку сходів вручну, переглянути стан датчиків, а також отримати повідомлення, якщо система зафіксувала рух або помилки. Окрім цього, для користувача системи доступна можливість перегляду журналу активностей (через Virtual Pins або webhooks).

Окрім цього, при такій організації системи можна імплементувати додаткові функції, які включають:

– налаштування часу роботи стрічки через слайдер у Blynk;

– нічний режим – автоматичне вмикання лише при низькому рівні освітлення за допомогою фоторезистора.

– виявлення аномалій – наприклад, якщо датчик постійно активний більше певного часу;

– статистика руху – рахунок активацій на день, відображення у додатку.

Існують комерційні комплекти систем підсвітки сходів від таких світових виробників, як Philips Hue, LIFX Z. Вони можуть бути інтегровані в системи розумного будинку. Однак їхня вартість значно перевищує собівартість розглянутих вище систем, а функціональність обмежена фірмовими платформами. Приклад впровадження комерційних рішень щодо освітлення сходин та поручнів сходової клітки показано на рис. 1.4.

Гнучкі Hue-стрічки акуратно заховані під сходами, забезпечують рівномірне підсвічування кожної сходини і створюють ефект м’якого світлового хвилювання. Такий варіант особливо популярний для модернізації інтер’єру завдяки вигідному поєднанню дизайну й функціональності.

Приклад з синьою RGB-підсвіткою під поручнями демонструє декоративне рішення для створення атмосферного освітлення, що адаптується під колірні сцени.

Решта прикладів ілюструє, як підсвічування працює через Hue Bridge, Hue-датчики руху або світлові стрічки Hue STRIP, що дозволяє створити автозапуск підсвітки при наближенні до сходів.

Рисунок 1.4 – Інтегровані у розумний будинок комерційні рішення підсвітки сходових кліток

Philips Hue надає готові та естетично привабливі рішення для сходів зі значною гнучкістю налаштувань, автоматизацією та інтеграцією в "розумний

дім". Модулі освітлення можна персоналізувати за кольором, плавністю і сценаріями, а датчики руху забезпечують безконтактне увімкнення.

Проаналізовані у даному розділі кваліфікаційної роботи системи управління підсвіткою мають і ряд недоліків та проблем, зокрема:

– потреба у стабільному підключенні до інтернету;

– високі вимоги до енергоефективності при автономній роботі;

– складність налаштування мережевих протоколів для недосвідчених користувачів;

– вразливість до кібератак за відсутності шифрування.

Проте, незважаючи на існуючі проблеми, системи віддаленого керування підсвітки сходів мають і перспективи розвитку. Застосування алгоритмів машинного навчання дозволить забезпечити адаптивність роботи підсвітки з врахуванням реакції на поведінкові патерни користувача.

Окрім цього, до перспектив розвитку таких систем належать:

- інтеграція з голосовими помічниками (Google Assistant, Alexa);
- застосування енергоефективних драйверів і самовідновних топологій з живленням;
- розширення функціональності на основі модульного підходу.

Системи підсвітки сходів з віддаленим моніторингом демонструють широкий спектр застосувань у сфері побутової автоматизації. Найбільш доцільним для сучасних проєктів є використання платформи ESP32 у поєднанні з мобільним застосунком Blynk або альтернативною хмарною службою. Це дозволяє реалізувати не лише базове освітлення, а й функції моніторингу, керування та аналітики. Подальші дослідження можуть бути спрямовані на забезпечення адаптивності та підвищення безпеки системи. У кваліфікаційній роботі пропонується проєкт системи віддаленого керування підсвіткою сходів через Telegram-бот із використанням адресної стрічки та контролера керування ESP32.

РОЗДІЛ 2 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ПІДСВІТКИ СХОДІВ З ВІДДАЛЕНИМ МОНІТОРИНГОМ

2.1 Архітектура системи віддаленого керування та моніторингу підсвіткою сходів

У роботі пропонується розглядати систему підсвітки сходів з віддаленим моніторингом у вигляді розподіленої інтелектуальної архітектури, яка орієнтована на управління адресною світлодіодною підсвіткою сходів у режимі реального часу. Вона базується на використанні мікроконтролера ESP32, Telegram-бота як віддаленого інтерфейсу керування, хмарної інфраструктури AWS для обробки запитів, збереження конфігурацій та двостороннього обміну даними через WebSocket.

Архітектура поєднує апаратну частину (edge-рівень) із хмарними обчисленнями (cloud), забезпечуючи масштабованість, безперервність роботи, віддалений моніторинг та управління. На рис. 2.1 показано загальну архітектуру системи.

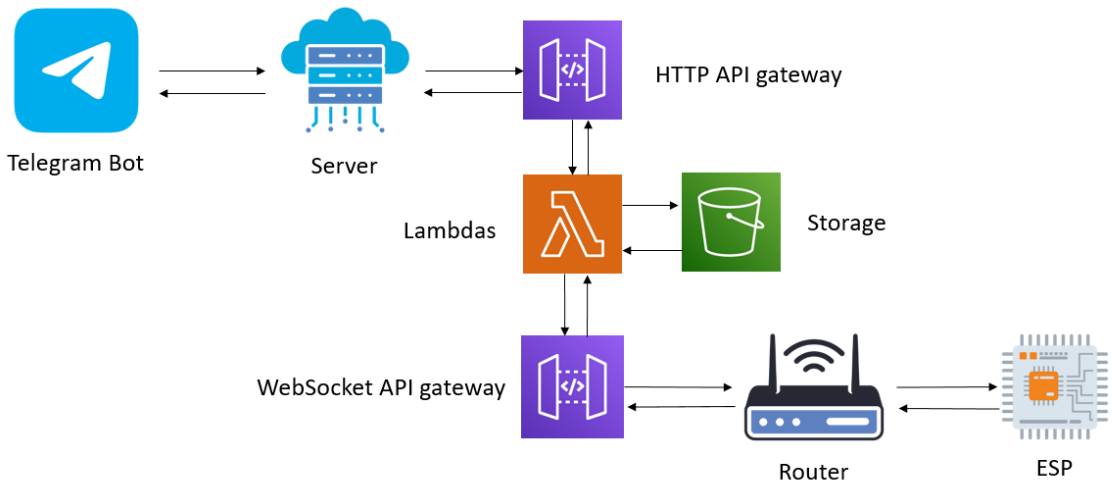

Рисунок 2.1 – Архітектура системи віддаленого моніторингу освітлення сходів

					КС КРБ 123.233.00.00 ПЗ	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Радій Р.І.			Проєктування комп'ютерної системи підсвітки сходів з віддаленим моніторингом	Літ. / Арк. 22 / Аркушів
Перевір.		Яцишин В.В.				
Реценз.						ТНТУ, каф. КС, гр. СІ-42
Н. Контр.		Тиш Є.В.				
Затверд.		Осухівська Г.М.				

Telegram-бот виступає як основний інтерфейс взаємодії з користувачем. За його допомогою можна надсилати команди вмикання/вимкнення підсвітки, змінювати кольори світлодіодів, обирати світлові ефекти (анімовані, статичні, реактивні). Окрім цього, даний месенджер забезпечує можливість отримувати поточний стан пристрою, зокрема, щодо режиму, кольорів, статусу підключення, а також ініціювати діагностику.

Використання Telegram API дозволяє зробити керування доступним із будь-якого пристрою з підключенням до Інтернету.

Центральний сервер відповідає за обробку повідомлень від Telegram-бота та переадресацію запитів через HTTP API Gateway. Він може працювати як проксі або вбудований логічний рівень авторизації, зберігання токенів, логування сесій користувачів.

Хмарна інфраструктура AWS, зокрема, HTTP API Gateway – це шлюз, що відповідає за обробку вхідних HTTP-запитів від Telegram-бота або інших зовнішніх джерел. Він виконує наступні функції:

– перевірка запиту;
– авторизація за токеном або ключем;
– маршрутизація на відповідну Lambda-функцію;
– контроль доступу;
– логування транзакцій.

API Gateway забезпечує масштабовану обробку запитів без необхідності розгортання власного бекенду.

Lambda-функції дозволяє реалізувати cервepless-логіка. Це обчислювальні модулі, які автоматично активуються при надходженні запиту. Lambda-функції реалізують логіку:

– обробки команд /on, /off, /color, /mode;
– валідації параметрів;
– підготовки повідомлення до ESP32 у відповідному форматі;
– формування відповідей у Telegram;
– взаємодії з WebSocket API Gateway для передачі команд на пристрій;
– логування дій користувача;

– керування збереженням налаштувань у S3.

Перевагою Lambda є можливість швидкого масштабування, відсутність потреби в постійній роботі сервера та сплата лише за час обробки.

Amazon S3 (сховище конфігурацій) відповідає за збереження стану пристрою та налаштувань користувача. Дані, які зберігає сховище при реалізації системи віддаленого моніторингу підсвітки сходів:

– останній активний режим підсвітки;
– вибраний колір або ефект;
– історія взаємодій;
– файли конфігурації для попередньо встановлених сценаріїв.

S3 є надійним, безпечним і доступним способом збереження параметрів, що дозволяє зберігати незалежність пристрою від локальних збоїв.

На відміну від HTTP, який є запитно-відповідною моделлю, WebSocket забезпечує постійне з’єднання між пристроєм (ESP32) та серверною частиною, дозволяючи миттєво надсилати команди на пристрій, отримувати зворотній зв’язок (наприклад, підтвердження виконання команди). Окрім цього, він може підтримувати канал зв’язку для моніторингу (heartbeat, keep-alive) та мінімізувати затримку у випадку частих подій (наприклад, швидка зміна режимів). Цей канал використовується для передачі даних між Lambda та ESP у реальному часі.

Роутер забезпечує локальне підключення ESP32 до мережі Інтернет, з можливістю отримання як вхідних WebSocket-повідомлень, так і запитів на оновлення прошивки (OTA), телеметрії тощо.

ESP32 – центральний виконавчий пристрій, який приймає команди з WebSocket-сервера, керує світлодіодною адресною стрічкою WS2812B, реагує на натискання локальних кнопок і відправляє зворотну інформацію (наприклад, підтвердження виконання команди). Також центральний пристрій керування здійснює контроль анімацій, кольорів, ефектів у реальному часі.

На ESP32 реалізовано прошивку з використанням FreeRTOS для паралельної обробки команд, управління стрічкою та взаємодії з мережею.

До переваг такої організації комп’ютерної системи підсвітки сходів з віддаленим моніторингом належить:

– розподіленість – відокремлення логіки функціонування (Lambda), зберігання конфігурацій і стану (S3) та інтерфейсу (Telegram) дозволяє забезпечити надійність і масштабованість системи.

– гнучкість – користувач може взаємодіяти з системою як локально, так і віддалено. Розширення функціоналу можливе без перепрошивання ESP32.

– надійність і безпека – AWS дозволяє використовувати авторизацію, захищені WebSocket-з'єднання, зберігання резервних копій налаштувань.

– масштабованість - один Telegram-бот може керувати декількома пристроями, кожен з яких має окремий канал WebSocket.

Запропонована на рис. 2.1 архітектура комп’ютерної системи може мати потенційні розширення. Це стосується можливості інтеграції з Google Assistant або Alexa, додавання сенсора освітленості для нічного режиму, автоматичне оновлення прошивки (OTA) через сервер та створення мобільного застосунку замість Telegram-інтерфейсу.

2.2 Пристрій керування комп’ютерною системою підсвітки сходів

Мікроконтролер ESP32 є сучасним високопродуктивним вбудованим обчислювальним модулем, розробленим компанією Espressif Systems. Він поєднує в собі обчислювальну потужність, низьке енергоспоживання та широкий спектр інтерфейсів, що робить його особливо привабливим для реалізації проєктів у сфері Інтернету речей (IoT), автоматизованого керування, смарт-будинків, інтелектуальних систем моніторингу та, зокрема, систем підсвітки сходів з функціями віддаленого доступу.

У центральній частині ESP32 знаходиться 32-бітний двоядерний процесор Tensilica Xtensa LX6, що працює на частоті до 240 МГц, що забезпечує достатній запас обчислювальної потужності для одночасного виконання декількох задач у режимі реального часу. На рис. 2.2 представлено зовнішній вигляд ESP32.

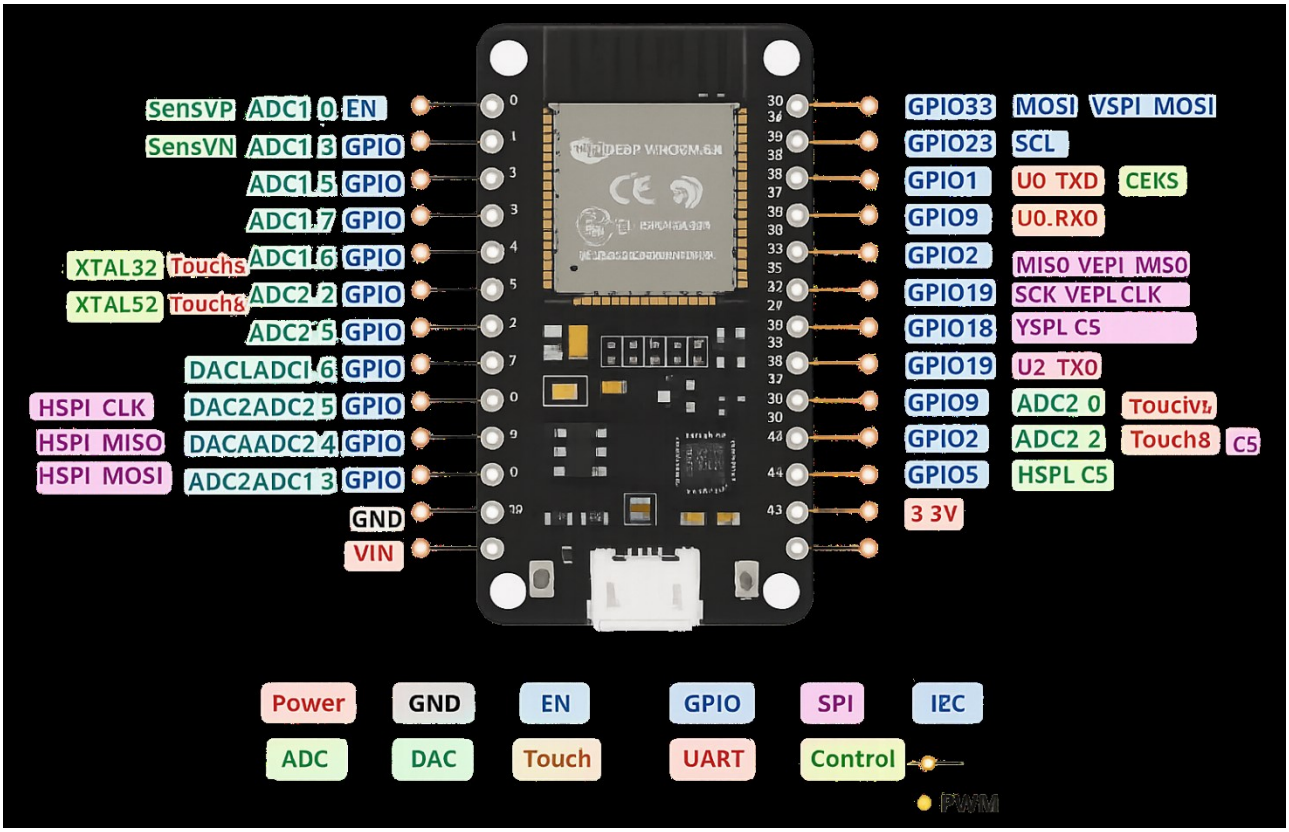

Рисунок 2.2 – Пристрій керування системи ESP32

Операційна система реального часу (наприклад, FreeRTOS), яка може бути розгорнута на ESP32, дозволяє ефективно керувати багатопоточними обчисленнями.

Окрім процесора, ESP32 оснащений вбудованим модулем Wi-Fi стандарту 802.11 b/g/n, а також Bluetooth. Це дозволяє пристрою забезпечувати бездротовий обмін даними з іншими пристроями, хмарними сервісами або мобільними застосунками без необхідності в додатковому мережевому обладнанні. Така інтеграція забезпечує ідеальні умови для систем, де потрібен як локальний контроль (наприклад, через кнопки), так і дистанційне керування (наприклад, через Telegram-бот або веб-інтерфейс).

ESP32 підтримує до 34 програмованих GPIO-виходів, які можуть бути використані для зчитування даних із сенсорів (руху, освітленості, присутності), а також для керування виконавчими механізмами, зокрема світлодіодними стрічками, реле, звуковими сигналами тощо. Завдяки вбудованим модулям

PWM, SPI, I²C, ADC, DAC, UART та CAN, ESP32 легко інтегрується в будь-яке цифрове або аналогове середовище.

У контексті автоматизованої підсвітки сходів ESP32 дозволяє реалізувати гнучку логіку освітлення, наприклад, поступове увімкнення світлодіодної стрічки при виявленні руху, зміна яскравості залежно від рівня освітлення в приміщенні або активація світла на основі часу доби. Застосування ESP32 також забезпечує можливість зберігати та обробляти дані про події, надсилати інформацію в хмарні сервіси (AWS, Blynk, Firebase) або локальні сервери для аналітики та логування.

Особливістю ESP32 є його надзвичайна гнучкість у енергоспоживанні: мікроконтролер підтримує кілька режимів енергозбереження, включаючи deep sleep, що особливо актуально для автономних систем з живленням від акумуляторів. Це дає змогу знизити енергоспоживання системи до мінімального рівня у випадках, коли пристрій не активний, зберігаючи при цьому здатність реагувати на події (наприклад, сигнал від сенсора руху).

Також важливою перевагою ESP32 є активна підтримка спільноти розробників, наявність великої кількості бібліотек, SDK (наприклад, ESP-IDF), відкритої документації та сумісність з популярними платформами розробки, такими як Arduino IDE, PlatformIO, MicroPython та NodeMCU. Це значно пришвидшує розробку прототипів і впровадження готових рішень.

Таким чином, ESP32 є серцем системи керування підсвіткою сходів, що дозволяє об'єднати апаратні та програмні компоненти в єдиний інтелектуальний комплекс з підтримкою автоматизації, дистанційного керування, обміну даними з мережею та високим рівнем адаптивності до вимог користувача.

Мікроконтролер ESP32 слугує основою системи на рівні «розумного» пристрою, безпосередньо підключеного до світлодіодної підсвітки сходів та до датчиків (наприклад, інфрачервоних датчиків руху на початку і кінці сходового прольоту). Прошивка, яка виконується на ESP32, відстежує сигнали датчиків (виявлення руху, рівень освітленості тощо) та в реальному часі керує ввімкненням і вимкненням LED-підсвітки. Вбудований Wi-Fi модуль ESP32 забезпечує з’єднання з локальною мережею через домашній маршрутизатор.

Маршрутизатор у цьому контексті виконує роль шлюзу до Інтернету, даючи змогу пристрою обмінюватися даними з хмарними сервісами. Для належної роботи системи потрібне стабільне підключення до інтернету (через Wi-Fi), аби ESP32 міг встановити зв'язок з хмарною інфраструктурою AWS. Таким чином, фізичний пристрій (ESP32) перебуває у локальній мережі, але має захищений канал зв'язку із сервером, що унеможливлює потребу прямого зовнішнього доступу до домашньої мережі.

При спрацюванні датчика або зміні стану освітлення ESP32 формує повідомлення з даними і надсилає його до хмари для подальшої обробки. У цій архітектурі для зв'язку використовується служба AWS API Gateway, яка надає інтерфейси API двох типів: HTTP (REST) та WebSocket. HTTP-інтерфейс призначений для обробки разових запитів або подій, зокрема, він використовується Telegram-сервісом для надсилання повідомлень бота (вхідні вебхуки) та може застосовуватися самим пристроєм для періодичної передачі даних.

WebSocket-інтерфейс API Gateway забезпечує постійне двостороннє з'єднання між ESP32 і хмарою, що особливо важливо для режиму реального часу. WebSocket API є двонаправленим: клієнт (пристрій) може надсилати повідомлення на сервер, і серверна логіка здатна самостійно ініціювати передачу даних або команд назад на клієнт. Така поведінка дозволяє хмарному сервісу «штовхати» (push) оновлення до пристрою без додаткових запитів, що дає змогу, наприклад, негайно передавати команду увімкнення світла зі сторони користувача. ESP32 підтримує постійне з'єднання через WebSocket, надсилаючи телеметрію (стан пристрою, події датчиків) і очікуючи можливі вхідні команди керування від сервера.

На стороні AWS усі ці взаємодії обробляються безсерверними функціями AWS Lambda. Кожен HTTP-запит або повідомлення по WebSocket, що надходить через API Gateway, автоматично передається у відповідну Lambda-функцію для обробки. AWS Lambda – це serverless-сервіс для виконання коду, який дозволяє запускати програмні функції у відповідь на події без необхідності керування інфраструктурою серверівscribd.com. У контексті системи підсвітки

сходів з віддаленим керуванням Lambda виконує роль «мозку» хмарної частини: саме тут реалізовано всю бізнес-логіку автоматизації. Отримавши дані від ESP32 (наприклад, повідомлення про те, що на сходах виявлено рух, або що світло було увімкнено), Lambda-функція може здійснити низку дій: зберегти отриману інформацію, проаналізувати її, прийняти рішення про подальші кроки. Наприклад, якщо вхідний сигнал свідчить про рух, Lambda може вирішити надіслати сповіщення користувачу через бот або записати цей інцидент до журналу. Якщо ж надійшла команда від користувача, Lambda інтерпретує її та формує відповідний вихідний сигнал до ESP32.

Безсерверна архітектура AWS робить ці функції надзвичайно масштабованими та швидкодіючими: середовище виконання автоматично масштабується під поточне навантаження, тому система може обробляти одночасно кілька подій без затримок. Значущою перевагою є й те, що розробникам не потрібно підтримувати постійно працюючі сервери – обчислювальні ресурси виділяються динамічно, тільки коли надходять події, що підвищує ефективність і надійність роботи.

2.3 Сенсори системи підсвітки сходів з віддаленим моніторингом

Інфрачервоний модуль HC-SR501 є одним із найбільш поширених засобів автоматичного виявлення руху, який ідеально підходить для вбудованих систем керування. Його робота ґрунтується на пасивній інфрачервоній технології, що дозволяє виявляти теплове випромінювання, яке випромінює людина або інші об’єкти з температурою, відмінною від фону. Основою конструкції виступає високочутливий сенсор LHI778, розміщений у напівсферичному лінзовому куполі типу Френеля, що фокусується на зоні огляду (рис. 2.3).

Даний сенсор має перевагу в низькому енергоспоживанні, що дозволяє застосовувати його в системах живлення від акумуляторів або резервних батарей. Пристрій ефективно використовується у проектах розумного дому, охоронних системах, а також у побутовій електроніці, включаючи системи

автоматичної підсвітки сходів. На основі рівня інфрачервоного випромінювання в полі зору, сенсор визначає появу об'єкта і активує цифровий вихід.

Рисунок 2.3 – Сенсор руху HC-SR501

Основні функціональні можливості HC-SR501 включають: регульований рівень чутливості, налаштовуваний час утримання сигналу після спрацювання (затримка вимкнення), а також перемикач режимів між одиничним спрацюванням і повторним (циклічним) тригером. Пристрої можуть працювати як у режимі одиничної активації, коли сигнал на виході з’являється лише на певний час, так і в режимі безперервного утримання, коли рух постійно підтримує вихід у стані логічної одиниці. Схема електрична принципова даного сенсора представлена на рис. 2.4.

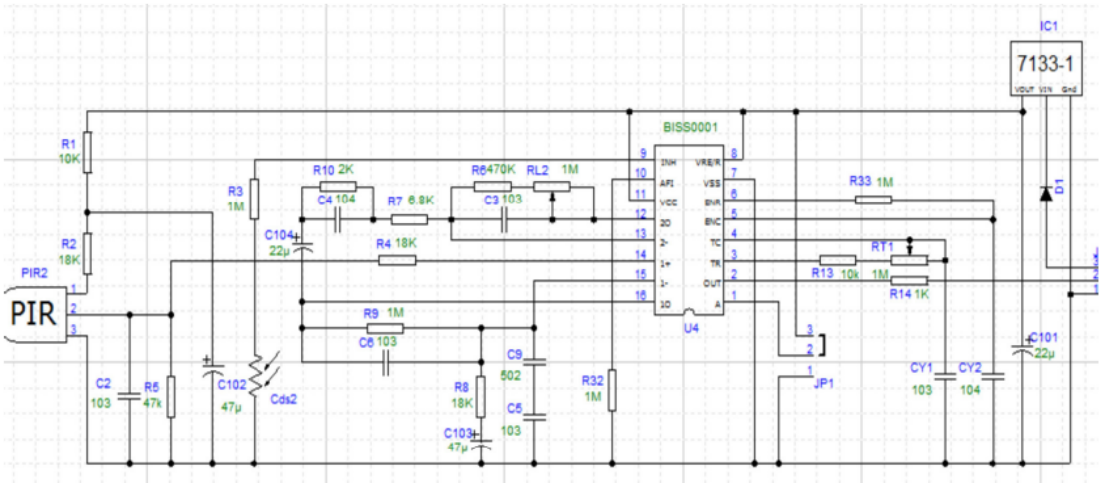

Рисунок 2.4 – Схема електрична принципова HC-SR501

Також передбачена температурна компенсація, що дозволяє сенсору залишатися ефективним навіть при підвищених температурах навколишнього середовища. Це особливо актуально для застосування в літній період, коли ефективність звичайних PIR-сенсорів може знижуватися.

Серед конструктивних особливостей слід відзначити: наявність потенціометрів для точного регулювання дальності спрацьовування (до 7 метрів) та тривалості затримки (до 5 хвилин), виводи живлення (VCC, GND) та цифровий вихід (OUT), що забезпечує TTL-сумісний сигнал для мікроконтролерів, таких як ESP32. Виводи HC-SR501 показано на рис. 2.5.

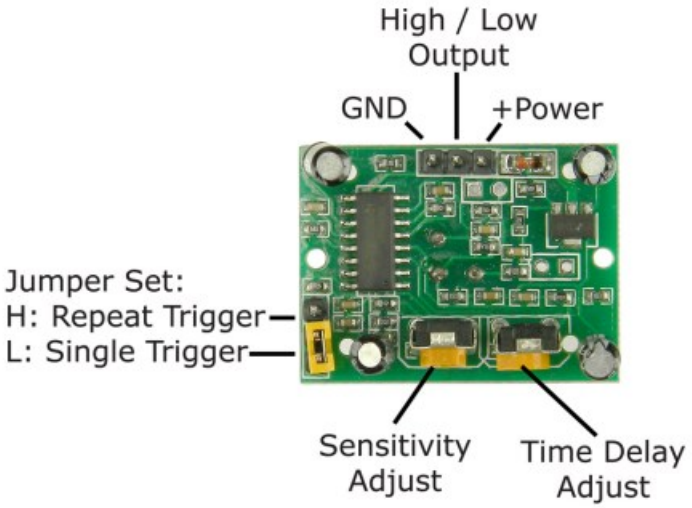

Рисунок 2.5 – Виводи HC-SR501

До основних технічних характеристик HC-SR501 належить:

– робоча напруга – від 5 до 20 В постійного струму.

– струм споживання – до 65 мА.

– цифровий вихід: рівень логічної одиниці — 3,3 В; нульовий рівень — 0 В.

– регульований час затримки – від 0,3 с до 300 с.

– час блокування між спрацюваннями – 0,2 с (за замовчуванням).

– два режими роботи – одиничне спрацювання (режим “L”) та повторне спрацювання (режим “H”).

– кут огляду – приблизно 120°.

– дальність виявлення – до 7 м.

– робочий температурний діапазон – від –15 до +70 °C.

– статичне споживання – <50 мкА.

Даний сенсор ідеально поєднується з мікроконтролером ESP32, який легко сприймає його цифровий вихідний сигнал, що дозволяє реалізувати надійні й енергоефективні рішення у системах підсвітки або безпеки. Схема підключення даного сенсора до ESP32 показана на рис. 2.6.

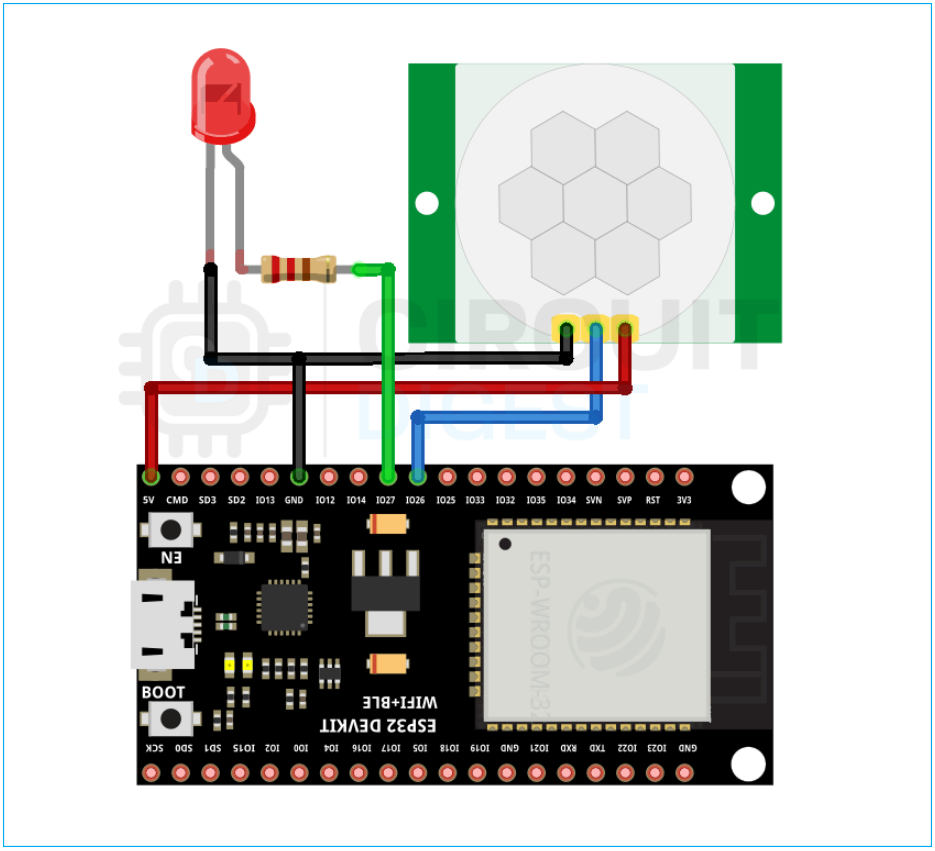

Рисунок 2.6 – Схема підключення HC-SR501 до ESP32

Сенсор присутності – це інтелектуальний компонент автоматизованих систем керування, який дає змогу виявляти наявність об’єкта (переважно людини) у визначеній зоні покриття. На відміну від стандартних сенсорів руху, які спрацьовують лише при динамічних змінах у полі зору, сенсор присутності здатен фіксувати навіть незначну активність або статичну присутність, наприклад, сидячої людини або незначні переміщення кінцівок. Саме ця особливість робить його незамінним елементом у системах освітлення, де

важливо зберігати світло ввімкненим, поки користувач фізично присутній, навіть без активного руху.

Сенсори присутності можуть ґрунтуватися на різних принципах роботи. Найчастіше в побутових і вбудованих рішеннях застосовуються два типи: інфрачервоні (IR) та мікрохвильові (Radar Doppler). У проектах на базі ESP32 перевагу часто надають мікрохвильовим датчикам, зокрема таким модулям як RCWL-0516, завдяки їхній здатності працювати крізь перешкоди (дерево, пластик, скло) і високій чутливості до навіть незначного переміщення (рис. 2.7).

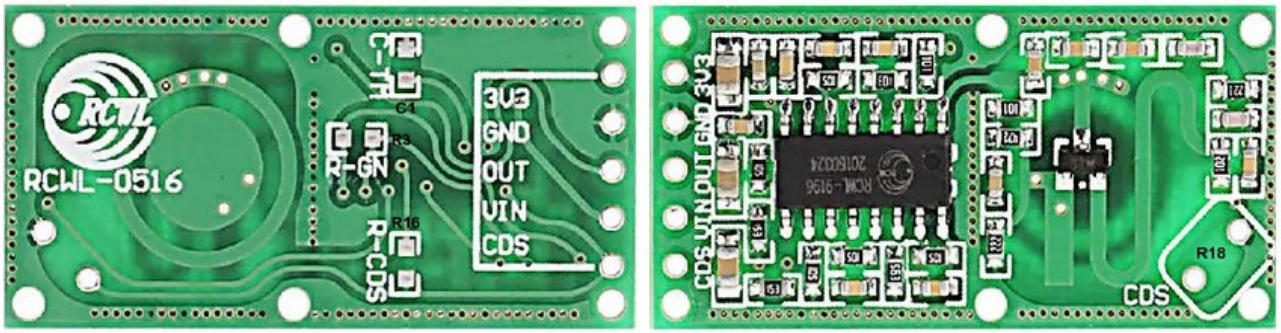

Рисунок 2.7 – Датчик присутності RCWL-0516

Принцип дії мікрохвильового сенсора базується на ефекті Доплера: сенсор генерує електромагнітні хвилі високої частоти (приблизно 3.2 ГГц) та аналізує їхнє відбиття від об'єктів у зоні огляду. Якщо об'єкт присутній і навіть незначно рухається, частота відбитого сигналу змінюється, що й фіксується мікросхемою сенсора як ознака присутності.

Модулі цього типу мають цифровий вихід, який можна безпосередньо зчитувати мікроконтролером ESP32, що значно спрощує інтеграцію в системи керування підсвіткою або сигналізацією. Вихідний сигнал тримається у високому логічному рівні, поки об'єкт перебуває в зоні дії сенсора, що дозволяє забезпечити безперервне ввімкнення освітлення або інших пристроїв керування без повторних спрацьовувань.

Завдяки невеликим розмірам, малій потужності споживання і високій точності виявлення, сенсор присутності ідеально підходить для розміщення в

інтер'єрі сходових маршів, коридорів або технічних приміщень. На рис. 2.8 показано схему підключення RCWL-0516 до ESP32.

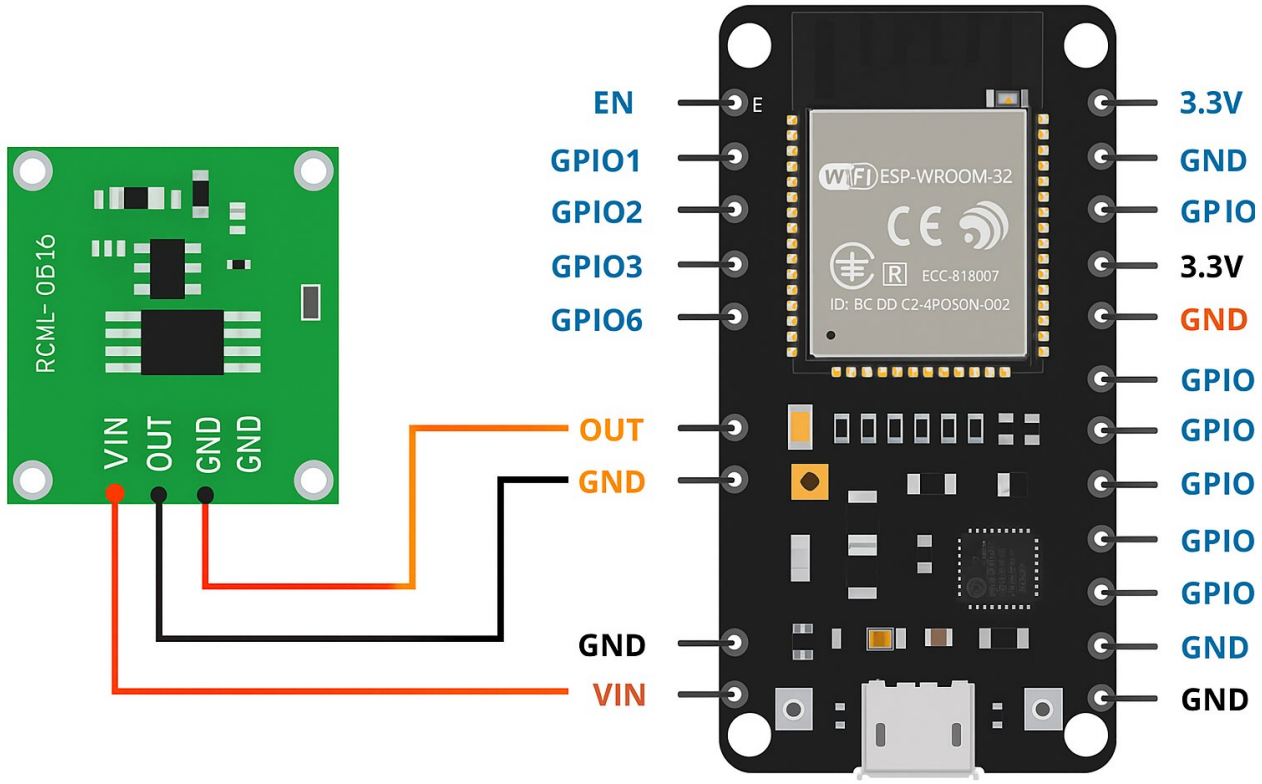

Рисунок 2.8 – Схема підключення RCWL-0516 до ESP32

Технічні характеристики модуля RCWL-0516) наступні:

– робоча напруга живлення: 4–28 В постійного струму;

– споживання струму: ~3 мА;

– робоча частота: ~3.2 ГГц (мікрохвильовий радар);

– час утримання вихідного сигналу: близько 2 секунд (може змінюватися в залежності від прошивки);

– дальність дії: 5–7 метрів;

– кут покриття: до 360° у горизонтальній площині;

– вихідний сигнал: логічний рівень TTL (3.3 В або 0 В);

– можливість роботи крізь нетеплопровідні матеріали (скло, дерево, пластик);

В умовах реалізації інтелектуальної системи підсвітки сходів та моніторингу, сенсор присутності дозволяє більш гнучко реагувати на поведінку користувача, залишаючи освітлення ввімкненим не лише при короткочасному русі, а й при тривалому перебуванні у контрольованій зоні. Це сприяє підвищенню зручності, зменшенню кількості помилкових вимкнень та загальному підвищенню рівня комфорту й безпеки в автоматизованій системі.

У системах автоматизованого керування освітленням, зокрема в інтелектуальних підсвітках сходів, сенсори освітленості відіграють ключову роль у визначенні поточного рівня зовнішнього світла та прийнятті рішення про ввімкнення чи вимкнення LED-стрічок. Для реалізації цієї функціональності можуть застосовуватись як прості аналогові компоненти, такі як фоторезистори, так і спеціалізовані цифрові сенсори, наприклад BH1750.

Фоторезистор є світлочутливим резистивним елементом, електричний опір якого змінюється залежно від рівня освітленості: чим інтенсивніше світло, тим менший опір. Для того, щоб інтегрувати LDR з мікроконтролером ESP32, використовується простий аналоговий дільник напруги, в якому фоторезистор з’єднаний послідовно з резистором фіксованого номіналу.

Напруга з центральної точки дільника подається на один із аналогових входів ESP32 (наприклад, GPIO34), де вона оцифровується за допомогою вбудованого АЦП. Таким чином, ESP32 отримує значення, що прямо пропорційне рівню навколишнього освітлення. Такий підхід є простим і бюджетним, хоча й має деякі обмеження щодо точності, стабільності вимірювання та залежності від температури.

Альтернативою LDR є цифровий сенсор освітленості BH1750 (рис. 2.9), який спеціалізовано розроблений для вимірювання інтенсивності світла в фізичних одиницях – люксах. Цей сенсор працює через інтерфейс I²C, що дозволяє зчитувати точні значення рівня освітлення без потреби в аналого-цифровому перетворенні.

Модуль BH1750 має вбудований фотодіод, підсилювач та АЦП, що забезпечують високу точність вимірювання (до 1 люкса) з мінімальними похибками. Завдяки стандартному протоколу I²C (підключення через SDA та

SCL, наприклад, GPIO21 та GPIO22 на ESP32), він легко інтегрується до системи. При цьому живлення може здійснюватися як від 3,3 В, так і від 5 В, що робить його сумісним з більшістю мікроконтролерів.

Рисунок 2.9 – Цифровий сенсор BH1750

BH1750 дозволяє програмно налаштовувати чутливість, а також має різні режими вимірювання — від швидкого до надточного. Завдяки цьому, він ідеально підходить для систем, де потрібно динамічно регулювати підсвітку в залежності від зовнішнього середовища.

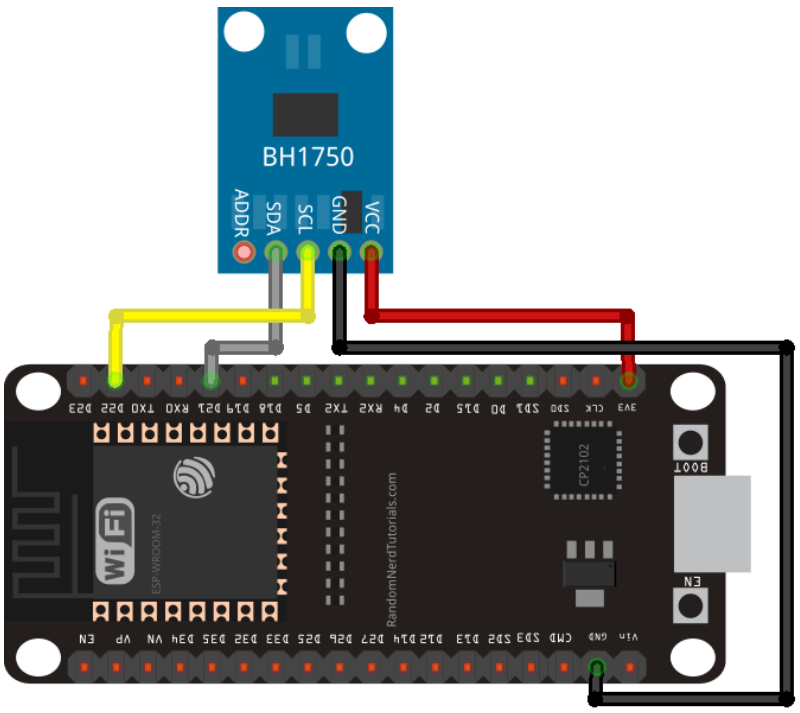

Рисунок 2.10 – Схема підключення BH1750 з ESP32

Фоторезистор із дільником напруги – це просте рішення для невибагливих застосунків, де важлива швидкість і вартість. Натомість BH1750 забезпечує більшу точність і надійність у складних умовах. У контексті «розумної» підсвітки сходів, сенсори освітленості допомагають вмикати підсвітку лише за умови недостатнього природного освітлення (наприклад, у темну пору доби або в затінених приміщеннях), що дозволяє зменшити споживання електроенергії та підвищити зручність користування.

Схема, що проілюстрована на рис.2.10, стандартна й надійна для більшості IoT-проєктів. Під’єднавши BH1750 до ESP32 так, як описано вище, можна отримати простий і точний спосіб для визначення рівня освітлення системи підсвітки сходів.

Типова схема підключення світлодіодних стрічок продемонстрована на рис. 2.11.

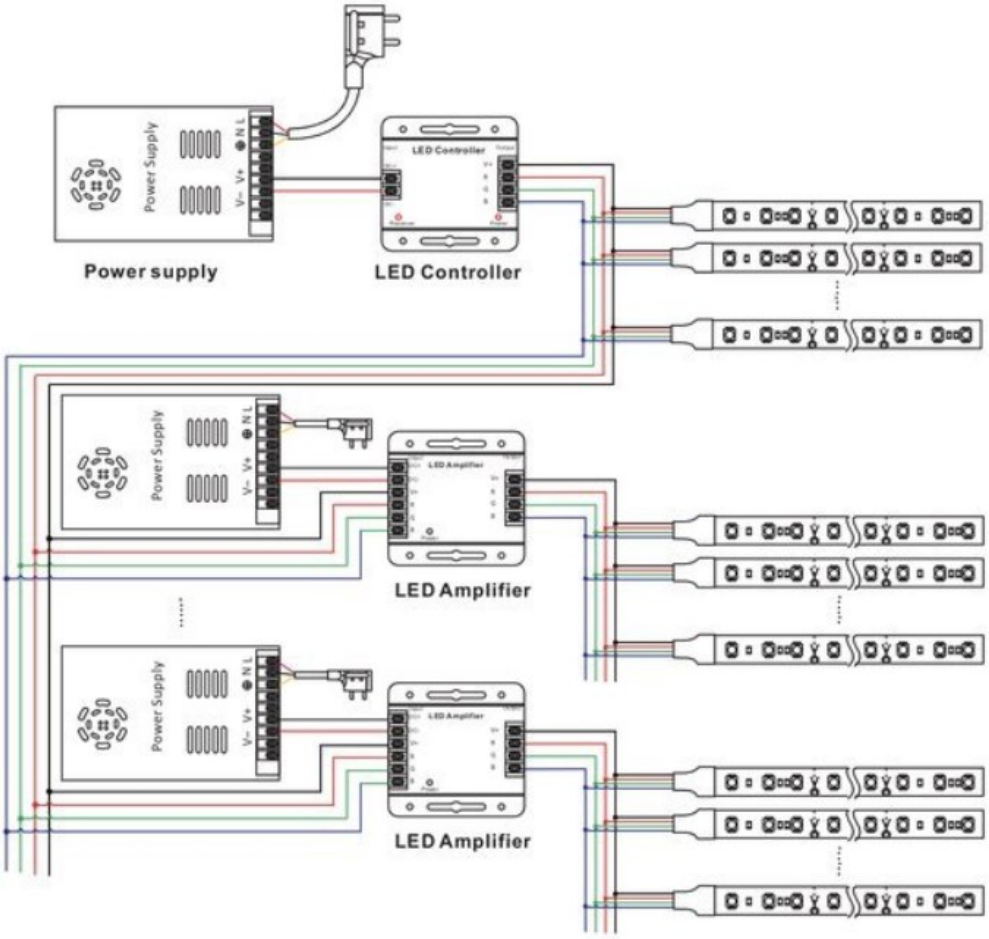

Рисунок .2.11 – Типова схема підключення світлодіодних стрічок

2.4 Схема підключення пристроїв комп’ютерної системи підсвітки сходів з віддаленим керуванням

Схема підключення компонентів комп’ютерної системи керування підсвіткою сходів на базі мікроконтролера ESP32 зображена на рис. 2.12. Вона включає сенсори для виявлення руху, присутності, рівня освітленості, а також кнопки для ручного керування та вибору режимів.

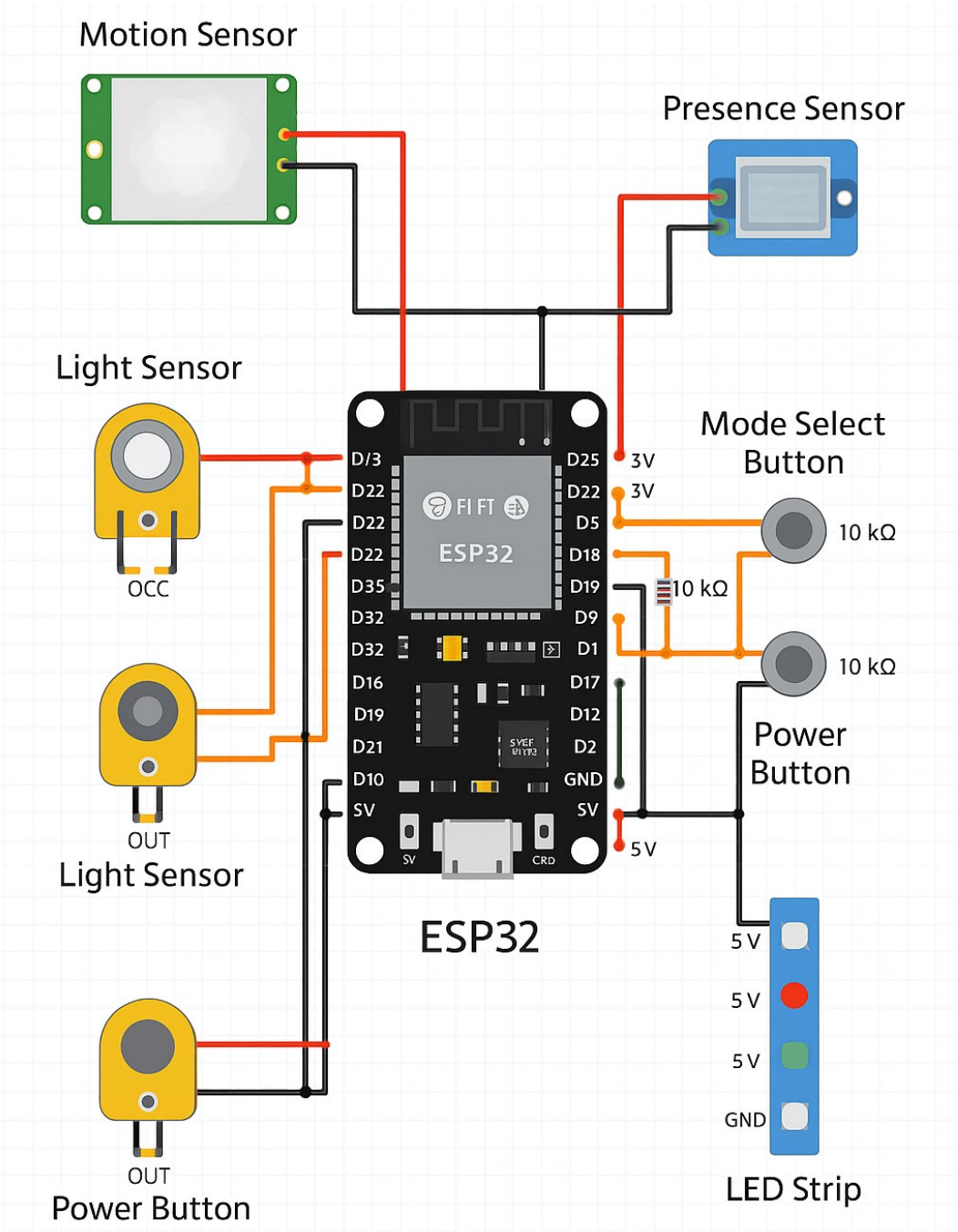

Рисунок 2.12 – Схема підключення компонентів системи підсвітки сходів

Мікроконтролер ESP32 виконує роль центрального елементу управління, з’єднаний з усіма сенсорами та виконавчими пристроями. Завдяки вбудованим Wi-Fi та Bluetooth модулям забезпечує можливість віддаленого моніторингу та керування підсвіткою.

Сенсор руху фіксує появу людини на початку чи в кінці сходів і підключається за схемою:

– VCC – до 3.3V ESP32;
– GND – до GND;
– OUT – до одного з цифрових входів ESP32 (GPIO14).

Сенсор присутності RCWL-0516 виявляє наявність людини у зоні сходів протягом більш тривалого часу. Схема його підключення до ESP32 наступна:

– VCC – 5V ESP32;
– GND – до GND;
– OUT – до вхідного GPIO ESP32 (GPIO27).

Сенсор освітленості BH1750 визначає рівень освітлення, щоб активувати підсвітку лише в темряві. Схема під’єднання до ESP32 наступна:

– SDA – до GPIO21;
– SCL – до GPIO22;
– VCC та GND – відповідно до живлення.

Кнопка увімкнення пристрою дає можливість ручного вмикання/вимикання всієї системи. Вона підключається до одного контакту GPIO (GPIO13) та до GND. Рекомендується застосування резистора 10 кОм або використати внутрішній pull-up ESP32.

Кнопка вибору режиму підсвітки забезпечують перемикання між кількома режимами підсвітки (наприклад: постійне світло, анімація, динамічна реакція на рух). Підключення аналогічне до кнопки увімкнення/вимкнення системи.

Адресна світлодіодна стрічка WS2812B є основним виконавчим елементом, що підсвічує сходи. Її схема підключення наступна:

– DIN – до одного з цифрових виходів ESP32 з підтримкою PWM (GPIO18)
– VCC – до зовнішнього джерела 5V (залежно від довжини стрічки);

– GND – спільна земля з ESP32;

Рекомендовано використати резистор (330–470 Ом) між GPIO ESP32 та DIN стрічки, а також конденсатор 1000 мкФ між VCC та GND для стабілізації напруги.

Живлення основних компонентів комп’ютерної системи передбачає використання джерела USB або через VIN (5V) для ESP32. LED-стрічка повинна обов’язково живитися від окремого джерела з достатнім струмом.

Таким чином, спроектовано комп’ютерну систему підсвітки сходів з віддаленим моніторингом та можливістю ручного управління. Коли один із сенсорів фіксує подію, наприклад, рух, ESP32 аналізує рівень освітленості, і при необхідності активує стрічку. Кнопки дозволяють змінювати режим або увімкнути підсвітку вручну. Якщо існує Wi-Fi підключення, то ESP32 може надсилати телеметрію до хмари чи Telegram-бота, або надсилати команди з бота до ESP32.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВІДДАЛЕНОГО МОНІТОРИНГУ КОМП'ЮТЕРНОЇ СИСТЕМИ ПІДСВІТКИ СХОДІВ

3.1 Алгоритм функціонування системи підсвітки на локальному рівні

Алгоритм роботи системи на локальному програмному рівні передбачає виконання сукупності послідовних кроків і перевірки ряду умов щодо режимів і стану, у якому перебувають компоненти. Після подачі живлення ESP32 виконує ініціалізацію вбудованих та зовнішніх модулів (датчиків, стрічки, кнопок), налаштування режимів GPIO. Після цього відбувається встановлення Wi-Fi-з'єднання та підключення до сервера моніторингу через MQTT/WebSocket або API. Наступний крок полягає в активації Telegram-бота .

Коли всі пристрої і з'єднання успішно активовані відбувається проце постійного зчитування даних із сенсорів. Контролер циклічно із заданою періодичністю виконує:

– вимірювання рівня освітленості через цифровий сенсор BH1750;

– зчитування даних з датчика руху HC-SR501 на предмет виявлення руху;

– визначення наявності людини сенсором присутності RCWL-0516);

– зчитування стану кнопок вмикання/перемикання режимів.

– аналіз поточного режиму – автоматичний, ручний, нічний тощо.

Логіка керування підсвіткою передбачає проведення аналізу стану датчиків і режимів функціонування системи підсвітки з віддаленим моніторингом.

Якщо виявлено рух або присутність людини і рівень освітлення менше порогового значення, то відбувається увімкнення світлодіодної стрічки, яка безпосередньо забезпечує освітлення сходів.

У разі активації нічного режиму буде застосовуватися приглушене світло. У випадку, коли рух не виявлено протягом певного часу, то автоматично здійснюється процедура вимкнення світла. На рис. 3.1 представлено загальний алгоритм роботи системи при управління на локальному рівні.

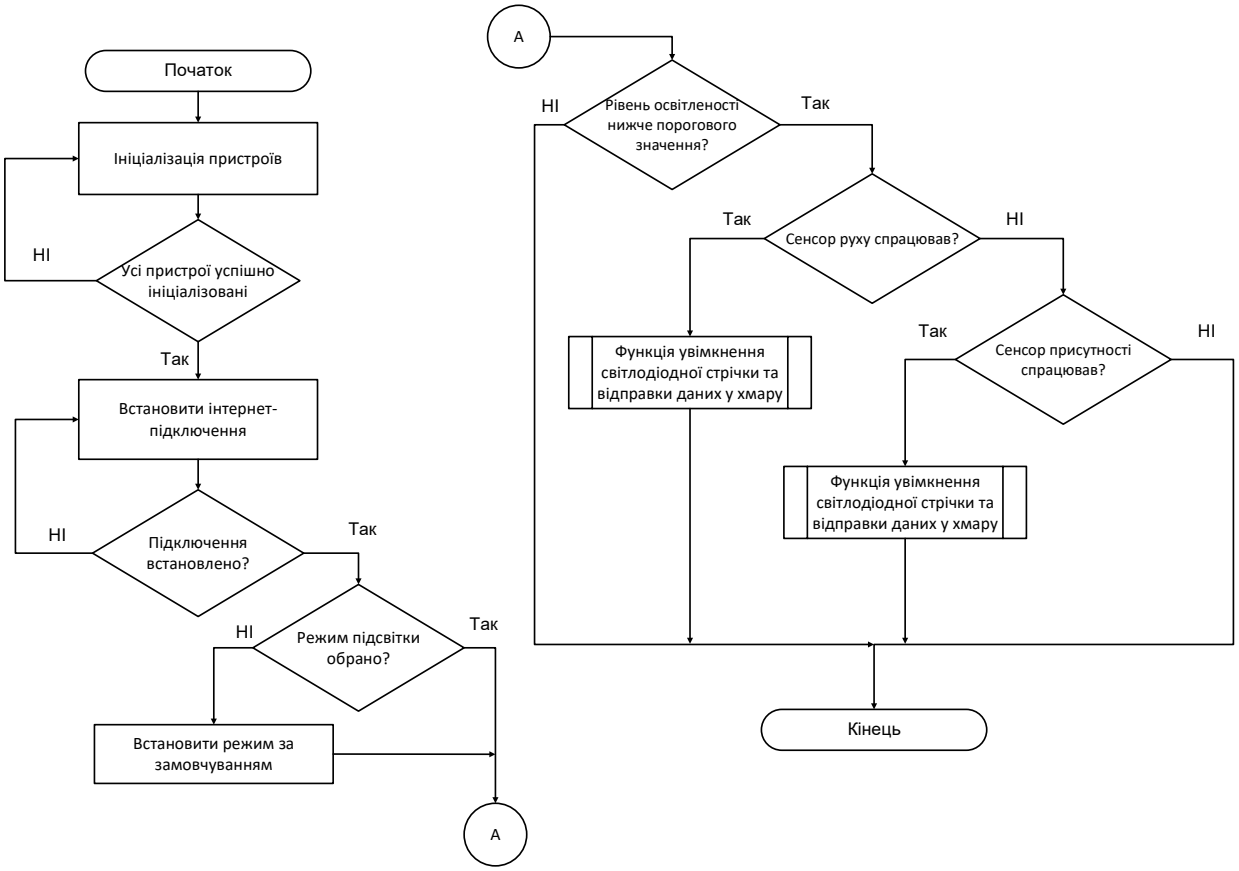

Рисунок 3.1 – Алгоритм роботи системи на локальному рівні

За допомогою кнопок локального управління користувач може увімкнути підсвітку вручну та перемикати режими: автоматичний / нічний / постійний / вимкнений.

Взаємодія з віддаленим сервером і Telegram-ботом відбувається при настанні подій, пов’язаних із спрацюванням датчиків та зміною режиму підсвітки. При цьому формується повідомлення і надсилається у Telegram, а дані логуються у хмарне сховище або на локальний сервер. Сервер може відправити команду на ESP32 (наприклад, увімкнути світло), отримати запит від користувача через Telegram-бота, виконати автоматичний сценарій (наприкад, активація режиму "Ніч").

Для організації захисту та безпечної роботи системи в перспективі можна передбачити звуковий сигнал у випадку несправності або довгого перебування людини у зоні дії сенсорів, аварійне відключення через фізичну кнопку, а також програмне перезавантаження через Telegram або апаратну кнопку.

3.2 Зберігання даних у хмарі (AWS S3)

Важливою складовою хмарної частини є сховище даних AWS S3 (Amazon Simple Storage Service). S3 використовується для надійного зберігання історичної та статичної інформації, пов’язаної з роботою системи. Це високодоступне та масштабоване об’єктне сховище, що надає безпечну та довготривалу платформу для зберігання і отримання даних.

У випадку системи підсвітки сходів в S3 можуть накопичуватися журнали подій (наприклад, часові мітки спрацьовування датчика руху та ввімкнення підсвітки), статистика використання, конфігураційні файли (режими роботи освітлення, налаштування яскравості тощо) або інші дані, важливі для моніторингу та аналізу.

S3 забезпечує централізований репозиторій таких файлів, гарантуючи їх збереженість і доступність для подальшої обробки. Завдяки тому, що AWS S3 автоматично реплікує дані у декількох зонах доступності, досягається висока стійкість до збоїв і відмов, а також глобальна доступність інформації. Це означає, що власник системи може в будь-який момент переглянути історію спрацювань чи змін налаштувань, а також що ці дані можуть бути використані іншими хмарними компонентами, наприклад, для аналітики або візуалізації.

Варто зазначити, що на базі S3 можна реалізувати і простий веб-інтерфейс: зберігши статичну веб-сторінку (HTML/JavaScript) у бакеті S3, можна відобразити дані IoT у графічному вигляді і мати доступ до цього інтерфейсу з будь-якого пристрою. Такий підхід ще більше розширює можливості мобільного моніторингу, дозволяючи користувачу спостерігати за станом підсвітки через браузер або додаток з привабливим інтерфейсом, що доповнює текстові повідомлення бота.

3.3 Telegram-бот як інтерфейс користувача

Для зручної взаємодії людини з системою інтегровано Telegram-бот, який виступає посередником між користувачем і хмарою. Telegram – це хмарний месенджер, доступний на смартфонах і комп'ютерах, що підтримує створення ботів – автоматизованих облікових записів, якими можна керувати через API. Спілкування з ботом відбувається у вигляді звичайного чату: користувач надсилає текстові команди або натискає спеціальні кнопки, а бот відповідає повідомленнями з інформацією чи підтвердженнями виконання команд.

Технологічно Telegram-бот реалізований таким чином, що всі повідомлення від користувача перенаправляються сервером Telegram на задану URL-адресу – вебхуку, котрим уцьому випадку є ендпойнт AWS API Gateway (HTTP). Іншими словами, Telegram «дзвонить» у хмару щоразу, коли бот отримує нове повідомлення від користувача. Цей запит приймається API Gateway і потрапляє до Lambda-функції, яка відповідає за обробку команд користувача. Подібна схема інтеграції дозволяє розгорнути бота на безсерверній інфраструктурі AWS та обробляти його запити лише коли вони фактично надходять, тобто не потрібно тримати постійний сервер для бота. Це є ефективним і економним рішенням.

Lambda-функція, яка реалізує логіку бота, виконує розбір вхідного повідомлення: з нього визначається, яка саме команда або запит надійшли. Наприклад, користувач може надіслати команду /lights_on для примусового ввімкнення підсвітки сходів. Отримавши такий запит через вебхук, хмарна функція перевіряє право доступу, щоб переконатися, що команда надійшла від авторизованого користувача і формує відповідне повідомлення для пристрою.

Через WebSocket-канал API Gateway ця команда негайно передається на ESP32, де її отримує прошивка пристрою. Контролер виконує команду – у даному разі вмикає світлодіодну стрічку на сходах і може надіслати зворотний статус про виконання. Коли ESP32 підтверджує виконання, наприклад, відправляє повідомлення про те, що світло ввімкнено успішно, Lambda-функція через Telegram Bot API надсилає користувачу відповідне повідомлення в чаті.

Взаємодія відбувається майже миттєво завдяки постійному з’єднанню: хмарний сервіс може в будь-який момент надіслати дані на клієнтський застосунок, тобто мобільний Telegram, без опитування з боку клієнта. Користувач також має змогу запитати стан системи. Для цього слугує команда, /state: після її надсилання ботом Lambda звертається до ESP32 через WebSocket із запитом надати актуальний статус. Отримавши запитувану інформацію від пристрою, хмара надсилає відповідь користувачу і бот генерує текстове повідомлення зі звітом про стан системи. Такий механізм забезпечує інтерактивний контроль, тобто користувач у будь-який момент може дізнатися, що відбувається з його системою, просто відправивши запит у чаті.

3.4 Сценарії взаємодії та сповіщення.

Архітектура системи підсвітки сходів з віддаленим моніторингом підтримує різні сценарії обміну даними між пристроєм та користувачем, охоплюючи як запити від користувача (керування), так і автоматичні сповіщення від пристрою.

По-перше, користувач може дистанційно керувати освітленням сходів. Через Telegram він надсилає команду боту, і вона майже миттєво виконується пристроєм. Важливо, що користувачу не потрібно знаходитися в локальній мережі – достатньо мати доступ до інтернету на смартфоні. Завдяки цьому рішення має глобальний характер: де б не перебував власник будинку, він завжди може отримати доступ до своєї системи освітлення.

По-друге, система здатна сама інформувати користувача про визначені події. Наприклад, якщо датчик руху на сходах спрацьовує у відсутності господаря, ESP32 сформує тривожне повідомлення і через хмарний шлюз ініціює його доставку користувачу. Lambda-функція, налаштована на обробку такого сигналу, викликає API Telegram та надсилає попередження на телефон власника. Таким чином, система поєднує функції безпеки сповіщення про неочікувану активність та автоматизації комфорту.

Користувач отримує повідомлення на смартфон миттєво, де б він не знаходився, за умови наявності інтернет-з'єднання. Це додає впевненості і зручності: власник може в реальному часі реагувати на ситуації або просто бути в курсі того, як і коли працює його система освітлення.

В якості користувацького інтерфейсу використовується переважно мобільний застосунок Telegram, оскільки бот інтегрується саме туди. Інтеграція з Telegram робить управління системою доступною і зрозумілою, оскільки користувач керує підсвіткою через звичайний чат, що не потребує окремих спеціалізованих додатків. З точки зору досвіду користувача це дуже зручно. Достатньо відкрити додаток Telegram на смартфоні, і можна як отримувати повідомлення від системи, так і надсилати їй команди. На практиці це означає, що використання системи не складніше, ніж спілкування з другом у месенджері. При цьому, якщо виникне потреба у розширеному інтерфейсі (графічному чи зі спеціальними контролами), архітектура дозволяє підключити і додатковий додаток. Наприклад, можна розробити окремий мобільний застосунок або веб-сторінку, що буде взаємодіяти з тим самим API Gateway, використовуючи HTTPS для запитів або під'єднуючись до WebSocket для отримання оновлень. Такий клієнтський застосунок може відображати візуальні індикатори стану підсвітки, статистику, надавати розширені налаштування і все це в реальному часі завдяки WebSocket-підключенню, яке дає змогу серверу безпосередньо надсилати дані на клієнт. У підсумку, користувач має кілька опцій для взаємодії: простий і легкий текстовий інтерфейс через бот або багатший графічний інтерфейс, якщо це необхідно, причому обидва можуть працювати паралельно і доповнювати один одного.

3.5 Роль хмарної інфраструктури та переваги архітектури.

Хмарна інфраструктура на базі AWS відіграє ключову роль у забезпеченні віддаленого доступу, обробки даних та інтеграції різнорідних компонентів системи. Вона виступає посередником між фізичним пристроєм і користувачем, беручи на себе всю складну логіку, зберігання інформації та взаємодію із

зовнішніми сервісами (зразок Telegram). Такий підхід має низку суттєвих переваг у контексті автоматизації, масштабованості, безпеки та гнучкості системи.

По-перше, автоматизація процесів стає значно простішою завдяки програмованості хмарної логіки. Використовуючи AWS Lambda, розробники можуть закласти складні алгоритми реагування на події, які виконуватимуться автоматично. Система здатна самостійно приймати рішення: вмикати підсвітку лише у темний час доби або після спрацювання кількох датчиків одночасно, надсилати повторні сповіщення, якщо рух фіксується протягом довшого часу, тощо. Усе це досягається без втручання людини. При цьому достатньо один раз запрограмувати потрібні правила. Хмара гарантує, що ці правила виконаються надійно, адже Lambda спрацює кожного разу при надходженні відповідної події.

Таким чином, досягається високий ступінь автоматизації побутового процесу освітлення: система сама керує підсвіткою за заданими умовами, водночас інформуючи власника про важливі події. Це підвищує комфорт і функціональність, оскільки сходи завжди освітлюються тоді, коли це потрібно, навіть якщо ніхто вручну не вмикає світло.

По-друге, масштабованість рішення практично не обмежена завдяки використанню хмари. AWS надає надзвичайно гнучку та масштабовану платформу, яка дозволяє легко адаптувати і розширювати систему за потреби, здатну обробляти дедалі більші обсяги даних чи збільшену кількість пристроїв. Якщо виникне потреба обслуговувати декілька сходових маршів, додати більше датчиків або навіть розгорнути такі системи підсвітки у багатьох будівлях архітектура легко з цим впорається. Достатньо підключити новий ESP32 з підсвіткою до тієї ж хмарної служби з окремим ідентифікатором, і він буде працювати під управлінням уже існуючої інфраструктури.

API Gateway і Lambda автоматично масштабуються для обробки зростаючої кількості запитів, а S3 без проблем вмістить додаткові дані. Таким чином, система, спроектована на серверless-принципах, може зрости від одного пристрою до сотень без суттєвої переробки архітектури. Важливо, що така масштабованість стосується і навантаження. Якщо раптово зросте частота подій,

наприклад, кілька користувачів одночасно надсилають команди або багато датчиків генерують сигнали, AWS Lambda запускатиметься паралельно в міру необхідності, майже без затримок для окремих викликів. В результаті користувачі не відчують погіршення якості сервісу і система залишатиметься швидкою.

По-третє, безпека є невід'ємною перевагою обраної архітектури. По-перше, всі з'єднання в системі є захищеними. ESP32 спілкується з AWS по протоколу HTTPS із шифруванням TLS, а взаємодія Telegram-бота з хмарою також відбувається через зашифровані вебхуки. Telegram, зі свого боку, шифрує повідомлення між клієнтом і сервером, тож канал від смартфона користувача до Lambda-функції захищений на всьому шляху. AWS особливо акцентує увагу на безпечній взаємодії IoT-пристроїв і хмари, надаючи механізми автентифікації, шифрування та керування доступом. У випадку системи віддаленого моніторингу за підсвіткою сходів можна використати токени або ключі API для авторизації запитів ESP32 до API Gateway, забезпечити, щоб тільки довірені пристрої надсилали дані. Аналогічно, Telegram-бот може бути налаштований так, щоб реагувати лише на повідомлення від певного користувача, використовуючи унікальний chat ID, що запобігає зловживанням. Дані, що зберігаються в S3, можуть бути шифровані як на рівні сервісу, так і додатково додатком. До них застосовуються політики доступу, які гарантують, що переглядати журнали або інші файли зможе лише авторизований власник. Відсутність прямих підключень до домашньої мережі (як-то відкритих портів) означає, що зловмисник не може безпосередньо достукатися до ESP32 – взаємодія йде тільки через керований шлюз AWS, який відфільтровує неприпустимі запити.

Таким чином, алгоритм роботи системи та архітектура поєднує в собі як фізичну безпеку (датчики і автоматичне освітлення відлякують непроханих гостей в темряві), так і кібербезпеку (захищені канали, автентифікація і авторизація на всіх рівнях). Власник системи може бути впевнений, що його дані і управління пристроями захищені сучасними засобами шифрування та контролю доступу.

По-четверте, гнучкість та модульність рішення дозволяють легко його розвивати. Якщо в майбутньому з’являться нові завдання, наприклад, інтеграція з іншими системами розумного дому, додаткові сценарії такі як реагування на пожежну сигналізацію чи інтеграція з голосовими помічниками, то це можна реалізувати, не змінюючи базової структури. Хмарна архітектура полегшує безшовне додавання нових функцій і підтримує майбутнє розширення системи. Достатньо дописати нову Lambda-функцію або розширити існуючу, додати новий ендпойнт в API Gateway або зберігати новий тип даних у S3 і система отримає новий шар функціональності.

Якщо, крім Telegram, виникне потреба мати інтеграцію з, припустимо, Google Home або Alexa, то відповідний хмарний сервіс можна підключити до тієї ж архітектури через додатковий AWS Lambda/API Gateway для обробки голосових команд без впливу на існуючий код для Telegram чи прошивку ESP32.

Така архітектурна гнучкість значною мірою досягається завдяки слабкому зв’язуванню компонентів: кожен елемент (пристрій, шлюз, обчислювальна функція, бот) взаємодіє через стандартизовані інтерфейси й не залежить від внутрішньої реалізації інших. Це спрощує підтримку і оновлення системи. Крім того, serverless-рішення AWS спрощують процес розгорнення оновлень.

У результаті поєднання апаратного пристрою ESP32, хмарної платформи AWS і месенджера Telegram утворюється цілісна комп’ютерна система для автоматизації освітлення, яка вирізняється надійністю та ефективністю. Вона демонструє переваги використання хмарних технологій в IoT-проєктах. З одного боку, досягається безперервна зв’язність і висока якість передачі даних між пристроєм та користувачем, з іншого – зберігається можливість гнучкого масштабування й вдосконалення.

Система підсвітки автоматично реагує на зміни середовища, забезпечуючи комфорт і безпеку, а користувач має зручні інструменти контролю через звичний мобільний інтерфейс. Завдяки хмарній архітектурі це рішення добре масштабується, захищене від несанкціонованого доступу та відмов окремих компонентів, і його легко адаптувати під нові вимоги. Така архітектура є прикладом сучасної практики IoT, оскільки вона поєднує автономну роботу

кінцевого пристрою мережі із сервісами хмари для досягнення оптимального балансу між локальним контролем і віддаленим моніторингом, гарантуючи високий рівень автоматизації, надійності та гнучкості системи в цілому.

3.6 Налаштування і програмна реалізація системи підсвітки сходів з віддаленим моніторингом

Перед початком реалізації програмної логіки функціонування системи підсвітки сходів з віддаленим моніторингом потрібно налаштувати параметри і створити необхідні сервіси у хмарному сервісі та месенджкрі Telegram. Створення бота є доволі простою процедурою, яка описана в офіційній документації. У результаті налаштування хмарних сервісів отримано розгорнуті API GateWay (рис. 3.2.), HTTP API для зв'язку з телеграм ботом (рис. 3.3) та WebSocket API для зв'язку з мікроконтролером (рис. 3.4).

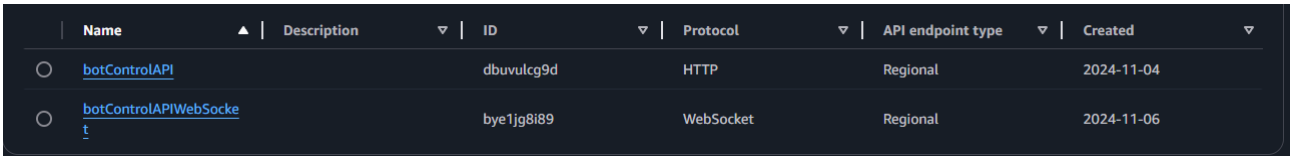

Рисунок 3.2 – Розгорнуті API Gateway

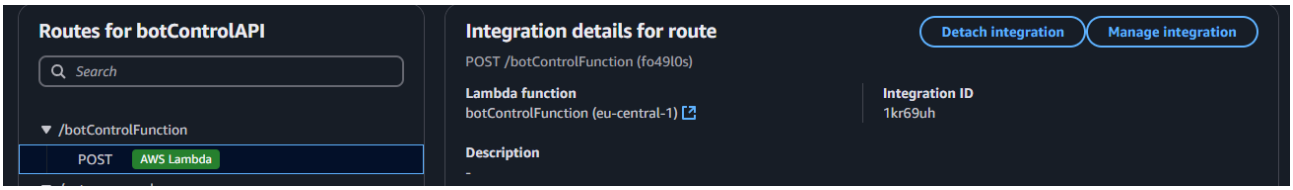

Рисунок 3.3 – HTTP API для зв'зяку з телеграм ботом:

WebSocket API для зв'зяку з мікроконтролером:

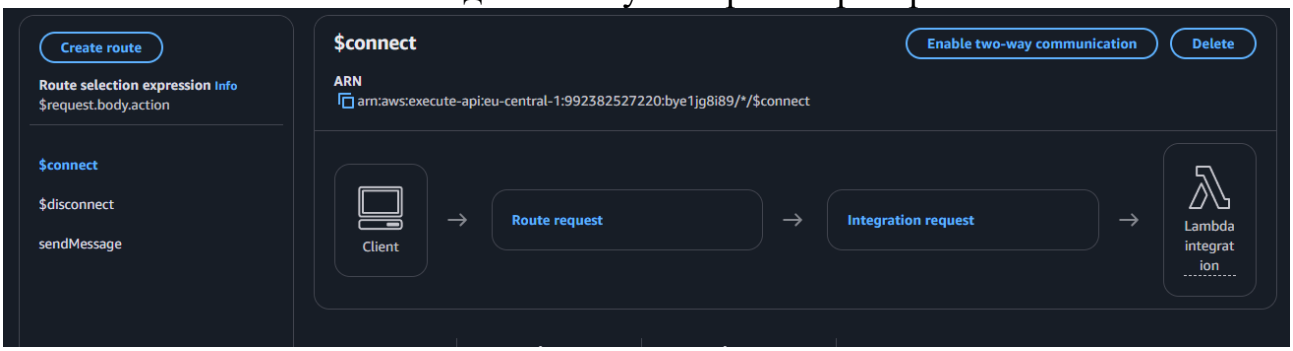

Рисунок 3.3 – WebSocket API для зв'язку з мікроконтролером:

Фрагмент програмного коду контролю мікроконтролера ESP32, який реалізує складну керовану систему підсвітки сходів, продемонстрований на рис. 3.4.

```
#include <Adafruit_NeoPixel.h>
#include <SPIFFS.h>
#include <WiFiManager.h>
#include <WebSocketsClient.h>
#include <ArduinoJson.h>

//#define DEBUG

#ifdef DEBUG
  #define LOG(text) Serial.println(text);
#else
  #define LOG(text)
#endif

#define PIXEL_PIN 27

#define NUMPIXELS 60

#define BTN_WIFI 4
#define BTN_MODE 18
#define BTN_COLOR 19
#define BTN_POWER 21
#define BTN_R 33
#define BTN_G 32
#define BTN_B 35

#define websocketServer "bye1jg8i89.execute-api.eu-central-1.amazonaws.com"

#define getChipId() ((uint32_t)ESP.getEfuseMac())

#define saveDataFileName "/data.json"

SemaphoreHandle_t xMutexColor;
SemaphoreHandle_t xMutexStates;
```

Рисунок 3.4 – Фрагмент коду контролю ESP32

При реалізації функцій контролю мікроконтролера ESP32 використано такі ключові компоненти та бібліотеки:

– Adafruit_NeoPixel – забезпечує керування WS2812-сумісною світлодіодною стрічкою;

– SPIFFS – дає змогу зберігати і зчитувати параметри кольорів, режимів, стану живлення;

– WiFiManager – забезпечує налаштування Wi-Fi через точку доступу;

– WebSocketsClient – встановлює з’єднання з сервером через WebSocket;

– ArduinoJson — виконує обробку вхідних/вихідних JSON-команд;

– FreeRTOS – дозволяє організувати багатозадачність з xTaskCreatePinnedToCore.

Визначення функціональності системи керування та моніторингу підсвіткою сходів передбачає програмне оголошення кнопок управління:

– BTN_WIFI – відкриває точку доступу для підключення Wi-Fi;

– BTN_POWER – вмикає/вимикає стрічку;

– BTN_MODE – переключає режим підсвітки;

– BTN_COLORa – входить у режим зміни кольору;

– BTN_R, BTN_G, BTN_B – змінюють значення R/G/B.

Програмно визначено також сценарії освітлення, зокрема, в залежності від значення змінної newState, викликаються ефекти:

– статичне світло;

– Pong-анімація;

– імітація завантаження (load);

– випадкове підсвічування.;

– «шахматка», тобто чергування двох кольорів;

– анімація з двома кольорами.

Збереження/завантаження стану перебачає збергіання кольору, режиму, chatId, стан повідомлення серверу у /data.json. Логіка хмарної інтеграції реалізована таким чином, що кожна команда з WebSocket аналізується і виконується, зокрема, зміна кольору, вмикання/вимикання, зміна режиму, а повідомлення надсилаються назад на сервер через WebSocket. FreeRTOS задачі

полягають в обслуговуванні WebSocket-з'єднання і Wi-Fi, виконання освітлювальних ефектів та реагування на натискання кнопок і виклик відповідні функції. Використання xSemaphoreTake/xSemaphoreGive гарантує, що кольори та стан не будуть зруйновані одночасним доступом. До загальних переваг такої реалізації належать повноцінне локальне керування, веб-керування через WebSocket, гнучка система ефектів, резервне збереження стану після перезавантаження, зміна кольору двома наборами RGB. Увесь програмний код контролю мікроконтролера ESP32 представлено у додатку Б. На рис. 3.5 показано код для підключення мікроконтролера.

```
import json
import boto3
from botocore.exceptions import ClientError

s3 = boto3.client('s3')
bucket_name = "esp-connection"

def lambda_handler(event, context):
    connection_id = event["requestContext"]["connectionId"]
    devices = []
    devices_states = {}
    default_device_state = {}


    query_params = event.get("queryStringParameters", {})
    device_id = query_params.get("device_id", "unknown_device")
    chat_id = query_params.get("chat_id")
    device_state = query_params.get("state")
    color_r = query_params.get("r", -1)
    color_g = query_params.get("g", -1)
    color_b = query_params.get("b", -1)

    print(f"Device id : {device_id}")
    print(f"Chat id : {chat_id}")
    print(f"State : {device_state}")
    print(f"Color : R={color_r}, G={color_g}, B={color_b}")

    if not chat_id:
        print("Chat id missing!")

        return {
            "statuCode" : 200
        }

    print("Pass chat check!")
```

Рисунок 3.5 – Фрагмент коду підключення ES

Код, представлений на рис.3.5 є реалізацією AWS Lambda-функції, яка виконує обробку HTTP-запитів WebSocket-з’єднання пристроїв (наприклад, ESP32) з хмарним сервером через API Gateway. Основна мета – керування підключеннями, зберігання станів пристроїв і кольорів у S3, а також обслуговування запитів від ESP-контролерів у режимі реального часу.

Даний код призначений для зберігання у S3 інформації про підключені пристрої ESP, їхні device_id, state та color, веденння обліку WebSocket-з’єднань (connectionId) та підтримки багатопотокового з'єднання в одному чаті.

Особливостями реалізації є застосування WebSocket-з’єднання, яке Зчитує connectionId для управління підключеннями. Хмарне сховище S3використовується як NoSQL-структура для зберігання даних. Для організації безпеки та обробки помилок інтегровано ClientError, Exception. Окрім цього, у коді імплементовано динамічне оновлення пристроїв, що додає або оновлює пристрої у конфігурації чату chat_id. Ідентифікатори. кольору та стану світлодіодної стрічки передаються як параметри і зберігаються. Типовий вміст S3-файлів проілюстровано на рис. 3.6 та рис. 3.7.


```
{
  "chat_id": "123456",
  "devices": ["esp_001", "esp_002"],
  "devicesStates": {
    "esp_001": {
      "state": "3",
      "color": { "r": "255", "g": "0", "b": "0" }
    }
  }
}
```


Рисунок 3.6 – Вміст /chats/{chat_id}.json

```
{
  "connectionId": "abcd1234",
  "device_id": "esp_001"
}
```

Рисунок 3.7 – Вміст /connections/{device_id}.json

Для програмного відключення мікроконтролера реалізовано функцію, яка представлена на рис. 3.8.

```
import json
import boto3

s3 = boto3.client('s3')
bucket_name = "esp-connection"

def lambda_handler(event, context):
    connection_id = event["requestContext"]["connectionId"]

    response = s3.list_objects_v2(Bucket = bucket_name, Prefix = "connections/")
    for obj in response.get("Contents", []):
        file_content = s3.get_object(Bucket = bucket_name, Key = obj["Key"])["Body"].read().decode('utf-8')
        data = json.loads(file_content)
        if data.get("connectionId") == connection_id:
            s3.delete_object(Bucket=bucket_name, Key=obj["Key"])
            break

    return {
        "statusCode": 200,
        "body": json.dumps({"message": "Disconnected"})
    }
```

Рисунок 3.8 – Функція відключення мікроконтролера

Фрагмент функції botControlFunction реалізовано для забезпечення зв’язку з телеграм ботом та передачі команди на мікроконтролер. Його проілюстровано на рис. 3.9. Цей скрипт реалізує AWS Lambda-функцію для керування пристроями ESP32 через Telegram-бот в рамках системи «розумного» освітлення сходів. Взаємодія базується на обміні командами через WebSocket API, з використанням хмарного сховища S3, AWS API Gateway і Telegram API.

У програмному коді (рис. 3.7) імплементовано функцію, яка дозволяє користувачеві надсилати Telegram-команди на зміну стану пристроїв.

Користувач може виконувати такі функції:

– вмикати/вимикати освітлення сходів;
– обирати режими світлових ефектів;
– задавати колір RGB;
– переглядати список підключених пристроїв;
– видаляти пристрої;
– запитувати поточний стан усіх пристроїв.

```python
import json
import requests
import boto3
from botocore.exceptions import ClientError
import enum

class commandCodes(enum.Enum):
    OFF = 1
    ON = 2
    STATIC = 3
    PONG = 4
    BREATHING = 5
    FIRE = 6
    START = 100
    SHOW_DEVICES = 101
    REMOVE_DEVICE = 102
    GET_STATES = 103
    ERROR = -1

s3 = boto3.client('s3')
bucket_name = "esp-connection"

BOT_TOKEN = "7751291125:AAHtmRtTAP8GdMIomz4-g06aTkRgD3q6gCs"

url = f'https://api.telegram.org/bot{BOT_TOKEN}/sendMessage'

def command_handler(message) -> int:
    match message:
        case "/on":
            return commandCodes.ON.value
        case "/off":
            return commandCodes.OFF.value
        case "/static_light":
            return commandCodes.STATIC.value
        case "/pong_light":
            return commandCodes.PONG.value
        case "/breath_light":
```

Рисунок 3.9 – Фрагмент функції botControlFunction ()

Перелік команд, які можна використовувати у Telegram-боті для моніторингу та управління підсвіткою сходів представлено у табл. 3.1.

Таблиця 3.1 – Команд иTelegram-бота

Команда	Опис
`/on <device_id>`	Увімкнути пристрій
`/off <device_id>`	Вимкнути пристрій
`/static_light <device_id> R G B`	Увімкнути статичне світло вказаного кольору
`/pong_light <device_id>`	Візуальний ефект "pong"
`/breath_light <device_id>`	Дихаючий ефект світла
`/fire_light <device_id>`	Імітація полум'я
`/remove_device <device_id>`	Видалити пристрій із чату
`/show_devices`	Список пристроїв у чаті
`/get_states`	Стани пристроїв (on/off + RGB)
`/start`	Вивід ID чату

Інтеграція Telegram-бота виконана шляхом застосування таких компонентів:

– BOT_TOKEN – використовується для надсилання повідомлень через Telegram API;

– send_feedback(chat_id, reply) – відправляє відповідь користувачеві.

Для отримання повідомлень від мікроконтролера та відправки фідбеку користувачу застосовується модуль sendMessage, фрагмент програмного коду якого, показано на рис. 3.10.

```python
import json
import urllib3
import boto3
import requests

BOT_TOKEN = "7751291125:AAHtmRtTAP8GdMIomz4-g06aTkRgD3q6gCs"

s3 = boto3.client('s3')
bucket_name = "esp-connection"

url = f'https://api.telegram.org/bot{BOT_TOKEN}/sendMessage'

def lambda_handler(event, context):
    print("Event received:", event)

    chat_id = 0
    device_id = ""
    message = ""

    reply = ""

    state = -1

    color = {"r": 0, "g": 0, "b": 0}

    connectionId = event["requestContext"]["connectionId"]

    body = {}

    if "body" in event:
        body_content = event["body"]
        print("Raw body content:", body_content)

        try:
            if isinstance(body_content, str) and body_content.strip():
                body = json.loads(body_content)
            else:
```

Рисунок 3.10 – Фрагмент модуля sendMessage

Команди у Telegram-боті можна вводити вручну, наприклад: /static_light 1111111 180 32 76. У даному випадку:

– /static_light – режим для лампи;
– 180 – значення інтенсивності свічення для RGB R;
– 32 – значення для інтенсивності свічення RGB G;
– 76 – значення інтенсивності свічення для RGB B.

Команди: /static_light, /pong_light, /load_light, /random_light, /checkers_light та /two_colors – вимагають обов’язково ідентифікатор пристрою, який можна отримати за допомогою команди /show_devices.

Колір є опційним, тому можна просто залишити: /static_light 1111111. Даними командами можна передавати і значення для іншої палітри кольорів: /static_light 1111111 45 65 23 98 100 32 – «45 65 23» - значення для першої палітри, «98 100 32» - для другої.

Якщо при виконанні команди отримується повідомлення: «Waiting for response from <device id>», то пристрій або втратив з’єднання або не має живлення.

Якщо отримується повідомлення «Device is not connected», то це означає, що пристрій не є підключений до мережі або не має живлення. Команда «/get_states» дозволяє переглянути в якому стані та кольорі перебувають сітлодіоди.

Таким чином, розроблено та налаштовано програмне забезпечення, що використовується для управління та моніторингу підсвіткою сходів з використанням IoT-пристроїв, хмарних сервісів та месенджера Telegram.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Менеджмент безпеки

Правове забезпечення безпеки життєдіяльності в Україні орієнтовано на державну політику щодо забезпечення життєдіяльності населення у техногеннобезпечному й екологічному чистому світі [19-21].

Екологічно чистий світ можливий лише при відсутності загрози з боку природних об'єктів чи при умові недопущення виникнення джерел техногенної безпеки. Із зазначених позицій основне місце посідає законодавство у галузі регулювання відносин з охорони здоров'я людини та навколишнього середовища, безпеки в надзвичайних та повсякденних ситуаціях, тобто безпеки життєдіяльності. Ці відносини регулюються нормативними актами різної юридичної сили: конституцією, законами, урядовими підзаконними актами, галузевими інструкціями вимог і правил безпеки життєдіяльності та відповідними актами місцевих органів влади [20].

Суспільство і держава відповідальні перед сучасним і майбутніми поколіннями за рівень здоров'я і збереження генофонду народу України, забезпечують пріоритетність охорони здоров'я в діяльності держави, поліпшення умов праці, навчання, побуту і відпочинку населення, розв'язання екологічних проблем, вдосконалення медичної допомоги і запровадження здорового способу життя.

Об'єктом управління БЖД є стан умов, параметрів і норм життєдіяльності на визначеній території або об'єкті. Головний напрямок у керуванні БЖД – створення безпечних умов життєдіяльності на всіх стадіях повного циклу функціонування системи "людина – навколишнє середовище".

Ефективне виконання будь-яких завдань організації управління та нагляду за безпекою життєдіяльності базується на безумовному дотриманні відповідних принципів [20]:

– системності, що передбачає застосування єдиних взаємоузгодженихпідходів до правового регулювання взаємовідносин у сфері компетенції Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи, служби Цивільної оборони;

– плановості, який полягає виключно у плановому порядку створення нормативно-правових актів;

– спадкоємності, який передбачає повне використання раніше напрацьованих матеріалів з вітчизняного та зарубіжного досвіду;

– економічності, який передбачає обов'язкове економічне обґрунтування всіх нормативно-правових актів;

– ієрархічності, який передбачає одночасну узгоджену розробку пакета нормативно-правових актів на державному та місцевому рівнях.

Контроль за дотриманням законодавства щодо безпеки життєдіяльності в Україні здійснюють різні державні та громадські організації. Серед них державні органи загальної, спеціальної та галузевої компетенції. До першої групи органів належать Кабінет Міністрів, виконавчі комітети місцевих рад народних депутатів, місцеві адміністрації.

Управління, контроль та нагляд за безпечною життєдіяльністю в Україні здійснюють [20]:

– Кабінет Міністрів України;
– Національна рада з питань безпечної життєдіяльності населення;
– Державна служба гірничого нагляду та промислової безпеки;
– Державна служба з надзвичайних ситуацій України;
– Державна інспекція ядерного регулювання України;
– Державний департамент пожежної безпеки;
– Заклади санітарно-епідеміологічної служби МОЗ.

Кабінет Міністрів України забезпечує:

– реалізацію державної політики в галузі охорони праці;

– затверджує національну програму щодо поліпшення стану безпеки, гігієни праці і виробничого середовища;

– визначає функції міністерств, інших центральних органів державної виконавчої влади щодо створення безпечних і нешкідливих умов праці та нагляду за охороною праці;

– визначає порядок створення і використовування державного, галузевих і регіональних фондів охорони праці.

З метою координації діяльності органів державного управління безпекою громадян та охороною праці створюється Національна рада з питань безпечної життєдіяльності населення, яку очолює віце-прем'єр-міністр України.

Національна рада організовує свою діяльність на підставі Положення про національну раду з питань безпечної життєдіяльності населення, затвердженого постановою Кабінету Міністрів України від 15 вересня 1993 р. № 733.

Запобігання виникнення небезпечних та несприятливих умов життєдіяльності передбачає підготовку і реалізацію комплексу правових, соціально-економічних, політичних, організаційно-технічних, санітарно-гігієнічних і інших заходів, спрямованих на регулювання умов і параметрів норм життєдіяльності, проведення оцінки рівня ризику, своєчасне реагування на зміну умов життєдіяльності на основі даних моніторингу, експертизи, контролю і прогнозу і недопущення переростання цих змін у небезпечні та несприятливі умови.

Ліквідація негативних наслідків передбачає скоординовані дії всіх структурних органів системи управління БЖД по реалізації заздалегідь розроблених планів, уточнених в умовах конкретного характеру і рівня надзвичайної ситуації з метою виключення загрози здоров'ю і життю людей і надання невідкладної допомоги по-страждалим.

Основні завдання та функції державної системи управління:

– планування робіт;

– розробка, прийняття і відміна нормативних актів;

– професійний відбір;

– реєстрація і облік;

– експертиза;

– ліцензування та сертифікація;

– управління фондами;

– узгодження і видача дозволів, наукове забезпеченя, міжнародне співробітництво;

– забезпечення безпеки обладнання, будівель та споруд;

– забезпечення санітарно-гігієнічних умов праці, санітарно-побутового обслуговування, лікувально-профілактичного і медичного обслуговування;

– розслідування та облік нещасних випадків, захворювань, аварій;

– пропаганда культури безпеки.

4.2 Естетичне оформлення та ергономічне дослідження робочого місця оператора

Ергономічна організація робочого місця користувача ЕОМ враховує як специфіку діяльності, що виконується, так і забезпечує комфортні умови перебування людини.

Тому основними ергономічними завданнями щодо організації робочого місця є наступні [21]:

– забезпечення просторових параметрів робочого місця, які відповідають антропометричним характеристикам користувача;

– раціональне розташування елементів робочого місця відносно користувача на підставі поглибленого кількісного та якісного аналізу діяльності, яка виконується;

– оптимізацію умов робочого середовища.

На рис. 4.1 наведено робоче місце користувача ЕОМ та позначено основні ергономічні та просторові параметри його складових.

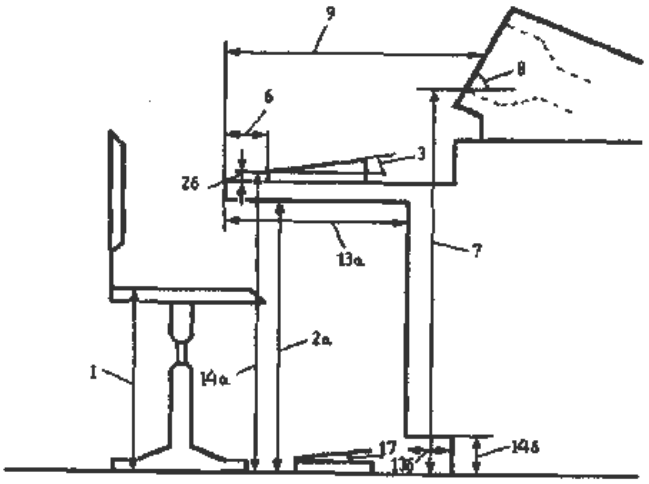

Рисунок 4.1 – Робоче місце користувача ЕОМ

Основні просторові параметри робочого місця користувача ЕОМ приведені у табл. 4.1.

Таблиця 4.1 – Просторові параметри робочого місця

Умовні позначення	Параметри	Спосіб вимірювання параметра	Значення параметра
1	Висота сидіння	Від підлоги до верхньої площини сидіння	400-500 мм
2	Висота клавіатури (від рівня підлоги)	Від підлоги до нижнього ряду клавіатури	600-700 мм
2а	Висота клавіатури (від рівня стола)	Від базової поверхні до нижнього ряду клавіатури	20 мм
3	Кут нахилу клавіатури	Від горизонтальної площини	$7\text{-}15^0$
4	Ширина основної клавіатури	Визначається оптимальною зоною моторного поля	До 400 мм
5	Глибина основної клавіатури	Визначається оптимальною зоною моторного поля	До 200 мм
6	Відстань від клавіатури до краю стола	Від переднього краю стола до клавіатури	Понад 80 - 100 мм

Продовж. табл. 4.1

Умовні позначен-ня	Параметри	Спосіб вимірювання параметра	Значення параметра
7	Висота екрана	Від підлоги до нижнього краю екрана	950-1050 мм
8	Кут нахилу екрана	Від вертикальної площини	15^0
9	Відстань від екрана до краю стола	Від переднього краю стола до екрана	500-700 мм
10	Висота поверхні для запису	Від підлоги	870-860 мм
11	Площа поверхні для запису	Визначається оптимальною зоною моторного поля	600 х 400 мм 900 х 600 мм
12	Кут нахилу поверхні для запису	Від горизонтальної площини	$0 – 10^o$
13	Глибина простору для ніг на рівні колін	Від переднього краю стола	Понад 400 мм
13а	Глибина простору для ніг на рівні	Від підлоги	Понад 600 мм
14	Висота простору для ніг на рівні колін	Від переднього краю стола	Понад 600 мм
14а	Висота простору для ніг на рівні ступень	Від підлоги	Понад 1 00 мм
15	Ширина простору для ніг на рівні колін		Понад 500 мм
15а	Ширина простору для ніг на рівні		Понад 250 мм
16	Висота підставки для ніг	Від підлоги до передньої частини підставки	50-130 мм
17	Кут нахилу підставки для ніг	Від горизонтальної площини	0-25
18	Глибина підставки для ніг	Від переднього краю підставки до її заднього краю	400 мм
19	Ширина підставки для ніг		300 мм
20	Пюпітр-підставка для документі в	Від горизонтальної площини	15 - 20°

В ході організації робочих місць на кожну ЕОМ виділяється площа, яка складає не менш, ніж 6 $м^2$, та об'єм, який становить не менш, ніж 20 $м^3$. Причому, зона, де розташовується робочий стіл, сервер або робоча станція, принтер, екран для графопроектора, займає відповідно 6-8 $м^2$. Висота приміщення не менша, ніж 4 м [13].

Робоче місце користувача ПК облаштоване одномісним столом та напівм'яким стільцем, висоту сидіння яких можна змінювати. Довжина стола користувача не менше 700 мм, ширина – забезпечує місце перед клавіатурою для розташування зошита або іншого приладдя. Поверхня стола має кут нахилу у межах $12\text{-}15^0$, лише іноді припустимою є її розташування у горизонтальній площині.

На робочому місці користувача ПК забезпечена відповідність висоти краю стола і стільця до росту та антропометричних особливостей організму користувачів.

Глибина простору для ніг під столом не менше 450 мм, а у випадку застосування високого стола та низького стільця і, отже, відсутності відповідності росту користувача конструктивним елементам робочого місця, використовується підставка для ніг, ширина якої становить – 350 мм, довжина – 400 мм, кут нахилу опорної поверхні – 15^o.

Столи з ЕОМ розміщено без розривів між ними, але при незначній кількості робочих столів з відеотерміналами перевагу варто віддавати розташуванню їх біля внутрішньої стіни.

Робота з комп'ютерною технікою вимагає обов'язкового дотримування правильної посадки. Користувач ЕОМ повинен сидіти прямо, з невеликим нахилом (до 5 – 7°) голови вперед, не сутулитися, спираючись нижніми краями лопаток на спинку стільця. Передпліччя повинні спиратися на поверхню стола, забезпечуючи зниження статичного напруження м'язів плечового поясу і рук, кути, що утворюються передпліччям і плечем, а також гомілкою і стегном, – складати не менш, ніж 90°.

Рівень очей припадає на центр екрана або на точку, яка розташована між верхньою та середньою третинами екрану, причому, лінія погляду є

перпендикулярною до площини екрана, а її відхилення у вертикальній площині – знаходитися у межах ±5–10^0. Оптимальний огляд у горизонтальній площині від центральної осі екрана у межах ±15–30°. Лише під час спостереження за інформацією, яка розміщена у найвіддаленіших ділянках екрану, кут огляду становить 40–45°.

Кут розглядання цифр та букв на екрані монітора не менше 20 кутових хвилин. Оптимальна відстань від очей до площини екрана монітора складає 600 –700 мм, допустима – не менше 500 мм. Розглядати інформацію на екрані з відстані менш, ніж 500 мм не рекомендується.

При проектуванні комп'ютерної системи враховані вимоги до ергономічної організації робочого місця користувачів ПК, що дозволяє підвищити працездатність та зменшити негативний вплив на їх здоров'я згідно ДСанПін 3.3.2.007 – 98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин».

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було спроєктовано та реалізовано комп’ютерну систему автоматичного керування підсвіткою сходів з підтримкою віддаленого моніторингу та налаштування.

Основним апаратним елементом системи є мікроконтролер ESP32, що забезпечує обробку даних з різних сенсорів — руху, присутності та освітленості, а також керування адресною світлодіодною стрічкою. За рахунок інтеграції з хмарними сервісами Amazon Web Services (AWS) та використання WebSocket-з’єднання, система дозволяє здійснювати віддалене керування освітленням за допомогою Telegram-бота, а також підтримує синхронізацію станів між пристроєм та сервером.

У межах реалізації було розроблено багатозадачне програмне забезпечення, яке включає обробку локального керування кнопками, збереження конфігурацій у файловій системі SPIFFS, обробку команд із хмари, зміни кольору підсвітки та перемикання між різними режимами візуалізації (статичне світло, ефекти "pong", "random", "checkers", "two colors" тощо).

Отримані результати демонструють, що запропонована система є гнучким і масштабованим рішенням для освітлення житлових приміщень, яке може бути адаптоване до індивідуальних потреб користувача. Вона не лише підвищує рівень комфорту і безпеки, а й сприяє економії електроенергії за рахунок автоматичного ввімкнення підсвітки лише за потреби.

Таким чином, поставлена мета була досягнута: створено функціональний, економічний та ефективний прототип комп’ютерної системи підсвітки сходів з можливістю віддаленого моніторингу, який має перспективу для подальшого вдосконалення та впровадження у повсякденне життя.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Романов Д.В., Осухівська Г.М., Паламар А.М. Система управління зовнішнім освітленням на основі Інтернету речей. Актуальні задачі сучасних технологій : збірник тез доповідей X міжнародної науково-практичної конференції молодих учених та студентів (Тернопіль, 24-25 листопада 2021 року), Тернопіль: ТНТУ, 2021. С. 120.

2. Романов Д.В., Осухівська Г.М., Паламар А.М. Функціональна схема системи керування зовнішнім освітленням на основі технології LoRa. Матеріали IX науково-технічної конференції "Інформаційні моделі, системи та технології" Тернопільського національного технічного університету імені Івана Пулюя (Тернопіль, 8–9 грудня 2021 року), Тернопіль: ТНТУ, 2021. С. 124

3. Лупенко С.А., Пасічник В.В., Тиш Є.В. Комп'ютерна логіка. Навчальний посібник. Львів: Видавництво «Магнолія 2006», 2024. 354 с.

4. Луцків А., Лупенко С., Пасічник В. Паралельні та розподільнені обчислення. Навчальний посібник. Львів: Видавництво «Магнолія 2006», 2024. 566 с.

5. Espressif Systems. ESP32 Series Datasheet. URL: https://www.espressif.com/en/products/socs/esp32 (дата звернення: 02.06.2025 р.).

6. Dogan I. Internet of Things with ESP32: Programming the ESP32 with Arduino IDE. Independently published, 2020. 385 p.

7. Adafruit NeoPixel Uberguide. URL: https://learn.adafruit.com/adafruit-neopixel-uberguide (дата звернення: 05.06.2025 р.).

8. AWS Documentation. Using AWS IoT Core and Lambda with ESP32. URL: https://docs.aws.amazon.com (дата звернення: 08.06.2025 р.)

9. Arduino UNO R4. Офіційна документація. URL: https://docs.arduino.cc/hardware/uno-r4 (дата звернення 04.06.2025 р.).

10. ESP32-CAM Camera Module. AI Thinker Documentation. URL: https://randomnerdtutorials.com/esp32-cam-video-streaming-web-server-camera-home-surveillance/ (дата звернення 03.06.2025 р.).

11. FreeRTOS. Official Real-Time Operating System Kernel. URL: https://www.freertos.org/ (дата звернення 04.06.2025 р.).

12. Yatsyshyn V., Pastukh O., Lutskiv A., Tsymbalistyy V. , Martsenko N. A Risks management method based on the quality requirements communication method in agile approaches. Information technologies: theoretical and applied problems, 2022. P. 1-10.

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

14. Yatsyshyn V., Pastukh O., Zharovskyi R., Shabliy N. Software tool for productivity metrics measure of relational database management system. Mathematical Modeling. No 1 (48). 2023. P. 7-17

15. Паламар М.І., Стрембіцький М.О., Паламар А.М. Проектування комп'ютеризованих вимірювальних систем і комплексів. Навчальний посібник. Тернопіль: ТНТУ. 2019. 150 с.

16. Жаровський Р.О., Тиш Є.В., Осухівська Г.М., Паламар А.М., Тиш Є.В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для розроблені у відповідності з освітньою програмою «Комп'ютерна інженерія» першого (бакалаврського) рівня вищої освіти за спеціальністю 123 «Комп'ютерна інженерія» галузі знань 12 «Інформаційні технології». Тернопіль, ТНТУ. 2024. 39 с.

17. Pastukh O., Yatsyshyn V. Brain-computer interaction neurointerface based on artificial intelligence and its parallel programming using high-performance calculation on cluster mobile devices. Scientific Journal of TNTU. Tern.: TNTU. 2023. Vol 112. No 4. P. 26–31.

18. Pastukh O., Yatsyshyn V. Development of software for neuromarketing based on artificial intelligence and data science using high-performance computing and parallel programming technologies. Scientific Journal of TNTU. Vol 113. No 1. 2024. pp. 143–149.

19. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». Київ. 2018.

20. Катренко Л.А., Катренко А.В. Охорона праці в галузі комп’ютинґу. Львів: Магнолія-2006. 2012. 544 с.

21. Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня „бакалавр”. Для студентів всіх форм навчання рівень вищої освіти перший (бакалаврський) / укл. : О. Я. Гурик , І. Б. Окіпний. Тернопіль : ТНТУ імені Івана Пулюя, 2021. 20 с.

22. Безпека життєдіяльності: навч. посіб. / Т.Є. Стиценко, Г.В. Пронюк, Н.М. Сердюк, І.І. Хондак. Харків: ХНРУЕ, 2018. 336 с.

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра комп’ютерних систем та мереж

“Затверджую”

Завідувач кафедри КС

_____ Осухівська Г.М.

“___” _____2025 р

КОМП’ЮТЕРНА СИСТЕМА ПІДСВІТКИ СХОДІВ З ВІДДАЛЕНИМ МОНІТОРИНГОМ

ТЕХНІЧНЕ ЗАВДАННЯ

на _*13*_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп’ютерна інженерія»

«УЗГОДЖЕНО»	«ВИКОНАВЕЦЬ»
Керівник кваліфікаційної роботи	Студент групи СІ-42
_____ к.т.н., доц. Яцишин В.В.	_____ Радій Р.І.
«____» _____ 2025 р.	«____» _____ 2025 р.

Тернопіль 2025

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Комп’ютерна система підсвітки сходів з віддаленим моніторингом».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.197.00.00

1.2 Виконавець

Студент групи СІ-42, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Радій Роман Іванович.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№4/7-53 від 27.01.2025 р.)

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 27.01.2025 р.

Плановий термін завершення виконання кваліфікаційної роботи – 18.06.2025 р.

1.5 Порядок оформлення та пред’явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи.

Попередній захист кваліфікаційної роботи відбувається при готовності роботи на 90% , наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Комп’ютерна система підсвітки сходів з віддаленим моніторингом повинна бути спроектована як інтерактивне комп’ютеризоване рішення, що забезпечує контроль і керування адресними світлодіодними стрічками. Систему пропонується реалізувати на базі мікроконтролера ESP32 з підтримкою Wi-Fi та Bluetooth, а також інтегрувати з популярним месенджером Telegram для реалізації інтуїтивно зрозумілого мобільного інтерфейсу.

Основною ціллю розробки є створення адаптивного освітлення, що реагує як на локальні дії користувача (через фізичні кнопки), так і на команди, надіслані з віддаленого пристрою. Ключовою особливістю є використання адресної світлодіодної стрічки WS2812B, кожним елементом якої можна програмно керувати. Це дозволяє реалізувати широкий спектр анімаційних ефектів, кольорових схем і режимів індикації.

На відміну від традиційних систем освітлення, проектована комп’ютерна повинна забезпечувати здатність адаптуватися до сценаріїв освітлення на основі команд користувача, зберігати конфігурацію у хмарному сховищі, обмінюватися інформацією з сервером у режимі реального часу за допомогою WebSocket-з’єднання.

Особливу увагу варто приділити доступності та надійності. Telegram-бот доступний у будь-який момент, а фізичні кнопки забезпечують дублювання функціоналу у випадку втрати мережевого з’єднання. Пристрій повинен також підтримувати функцію автоматичного перепідключення до Wi-Fi та збереження останнього активного стану освітлення, що зробить його стабільним та зручним у повсякденному використанні.

Таким чином, система призначена не лише для реалізації візуального комфорту чи декоративного підсвічування, але й для демонстрації практичного застосування IoT-підходів у побутовому середовищі, об'єднуючи апаратне забезпечення, програмну логіку та хмарну інтеграцію в єдину функціональну платформу. Вона також є масштабованою, із можливістю керування кількома пристроями з одного інтерфейсу.

2.2 Мета створення системи

Основна мета створення системи моніторингу освітленням сходів полягає у підвищенні гнучкості, зручності та функціональності сучасних освітлювальних рішень шляхом поєднання локального та віддаленого керування, інтеграції з мобільним інтерфейсом (Telegram-ботом) та забезпечення двостороннього моніторингу через хмарну інфраструктуру.

Проєкт орієнтований на реалізацію системи, яка дозволяє користувачеві змінювати параметри підсвітки на базі WS2812B у режимі реального часу, створювати індивідуальні сценарії освітлення, а також отримувати актуальний стан пристрою незалежно від фізичної близькості до нього.

Досягнення поставленої мети можливе через виконання наступних задач:

– проведення аналітичного огляду наявних рішень керування освітленням та систем з Telegram-інтеграцією;

– розробка структурної та функціональної архітектури комп’ютеризованої системи керування адресною стрічкою на базі ESP32;

– обґрунтування вибору апаратного забезпечення (контролер, кнопки, стрічка WS2812B, джерело живлення);

– побудова схеми підключення апаратних компонентів з урахуванням енергетичних та комунікаційних характеристик;

– реалізація програмного забезпечення, яке забезпечує обробку команд з Telegram, взаємодію з WebSocket-сервером та керування світлодіодами;

– проведення тестування готової системи, вимірювання затримки реакції, перевірка стабільності зв’язку з мережею та оцінка надійності роботи у різних сценаріях.

2.3 Характеристика об'єкту

Об’єктом дослідження у кваліфікаційній роботі є комп’ютерна система віддаленого моніторингу освітлення, що реалізується з використанням адресної світлодіодної стрічки WS2812B, мікроконтролера ESP32 та технологій інтернету речей (IoT). Система забезпечує як локальне, так і віддалене керування підсвіткою в режимі реального часу, що робить її універсальним рішенням для автоматизованого керування сценаріями освітлення.

Головною особливістю об'єкта є здатність виконувати інтерактивну зміну станів освітлення: колір, яскравість, режим за допомогою двох паралельних каналів управління, зокрема, фізичних кнопок, які підключені до контролера ESP32 і Telegram-бота, що працює через хмарну інфраструктуру (API Gateway + WebSocket + Lambda).

Система побудована на базі архітектури реального часу, яка використовує FreeRTOS для забезпечення паралельного виконання задач. Кожен світлодіод у

стрічці програмується індивідуально, що дозволяє реалізовувати складні анімаційні ефекти та сценарії з високою точністю.

Особливу увагу в системі приділено адаптивності освітлення завдяки адресності WS2812B, дистанційному контролю через Telegram-інтерфейс та двосторонній синхронізації з сервером за допомогою WebSocket.

У ситуації втрати з’єднання з мережею або сервером, система автоматично переходить у локальний режим керування без втрати поточних налаштувань. Крім того, всі основні дії супроводжуються зворотним повідомленням у Telegram, що дозволяє користувачу мати повний контроль над станом пристрою.

3 Вимоги до системи

3.1 Вимоги до системи в цілому

Комп’ютерна система підсвітки сходів з віддаленим моніторингом – це багатофункціональна IoT-система, що забезпечує керування освітленням на основі подій, пов’язаних із присутністю користувача, а також дозволяє здійснювати моніторинг та конфігурацію у режимі реального часу. Загальні вимоги до системи поділяються на функціональні та нефункціональні, що охоплюють як апаратні, так і програмні аспекти її роботи.

До ключових функціональних можливостей системи належать:

– автоматичне вмикання та вимикання світлодіодної підсвітки при виявленні руху на сходах;

– поступове включення світла знизу вгору або згори вниз залежно від напрямку руху користувача;

– локальне керування режимами підсвітки за допомогою фізичних кнопок (наприклад, вибір сценарію освітлення);

– можливість дистанційного керування підсвіткою через мобільний застосунок або месенджер (наприклад, Telegram-бот);

– збереження поточного режиму освітлення та його автоматичне відновлення після перезавантаження;

– надсилання стану пристрою на сервер моніторингу через WebSocket у режимі реального часу;

– підтримка декількох сценаріїв підсвітки (анімовані, статичні, нічний режим);

– можливість ручного керування системою при відсутності з’єднання з мережею Wi-Fi.

Програмне забезпечення системи повинно забезпечувати:

– обробку сигналів з датчиків руху (HC-SR501) та освітленості з мінімальною затримкою (до 1 с);

– реалізацію анімаційного сценарію підсвітки з урахуванням напряму руху користувача;

– передачу інформації про стан системи (увімкнення, активний режим, рівень освітлення) через WebSocket до хмарної платформи (наприклад, AWS);

– реакцію на запити з Telegram-бота в межах до 3 секунд;

– логування подій, що включає фіксацію часу активації, активного режиму, статусу датчиків та користувацьких дій;

– підтримку багатозадачності за допомогою операційної системи реального часу (наприклад, FreeRTOS на ESP32).

До нефункціональних характеристик системи належать:

– зручність та ергономіка реалізуються через користувацький інтерфейс Telegram-бота, який повинен бути інтуїтивно зрозумілим;

– надійність – система має функціонувати без збоїв при постійному циклі увімкнення/вимкнення протягом не менше ніж 72 годин;

– доступність – забезпечення дублювання основних функцій шляхом локального керування, яке повинно працювати незалежно від доступу до Інтернету;

– захищеність – передбачає використання авторизаційних токенів при взаємодії з Telegram-ботом, шифрування даних між пристроєм та сервером;

– масштабованість – можливість додавання нових сценаріїв освітлення, підключення додаткових сенсорів або розширення стрічки;

– енергоефективність – зменшення яскравості в нічний час, оптимізація тривалості підсвітки.

3.1.1 Вимоги до способів та засобів зв’язку між компонентами системи

У комп’ютеризованій системі автоматизованої підсвітки сходів передбачається використання єдиного багатофункціонального мікроконтролера ESP32, який виступає в ролі центрального керуючого вузла. Комунікація між окремими функціональними модулями системи – сенсорами руху, присутності, освітленості, а також керувальними кнопками та світлодіодною стрічкою – реалізується через провідні інтерфейси GPIO відповідно до призначення кожного компонента. Пристрій має бути здатен здійснювати обробку сигналів від датчиків у реальному часі, забезпечуючи активацію або деактивацію підсвітки в залежності від ситуацій у сходовому прольоті.

Для реалізації віддаленого моніторингу та керування системою мікроконтролер повинен підтримувати бездротове з’єднання з мережею Wi-Fi. Саме через цей канал забезпечується взаємодія з хмарними службами, зокрема передача телеметричних даних та отримання команд користувача за допомогою Telegram-бота. У системі застосовується гібридна модель зв’язку: провідникове підключення використовується для під’єднання фізичних сенсорів та виконавчих елементів, тоді як безпровідне з’єднання відповідає за обмін інформацією з сервером та іншими зовнішніми сервісами. Така архітектура гарантує гнучкість, розширюваність та мінімальну затримку при виконанні команд.

3.1.2 Вимоги по діагностуванню системи

Процес діагностування комп’ютеризованої системи підсвітки сходів із підтримкою віддаленого моніторингу має охоплювати як перевірку окремих апаратних компонентів, так і оцінку функціональності всієї системи в інтегрованому

режимі. Первинна перевірка виконується під час запуску пристрою (ініціалізації), коли здійснюється контроль наявності сенсорів руху, присутності, освітленості, кнопок керування, а також справності адресної світлодіодної стрічки. На цьому етапі ESP32 повинен зчитувати базові дані з кожного пристрою та формувати лог-файл, який може бути збережений у внутрішній пам’яті (SPIFFS) або переданий на хмарний сервер. За цими логами проводиться оцінка роботи кожного вузла – визначаються потенційні збої, недоступність або відсутність відповіді від датчика.

Особливу увагу слід приділяти перевірці справності безпровідного з’єднання через Wi-Fi, що дозволяє здійснювати діагностику у віддаленому режимі, а також відстежувати передачу команд і телеметрії між ESP32 та серверною частиною (наприклад, Telegram-ботом або AWS Lambda). У процесі роботи система повинна періодично надсилати звіт про свій стан, зокрема поточний режим роботи, активність кнопок, рівень освітленості та виявлення руху.

3.1.3 Перспективи розвитку, модернізація системи

Подальший розвиток комп’ютерної системи підсвітки сходів із функцією віддаленого моніторингу передбачає не лише вдосконалення поточної архітектури, але й інтеграцію нових функцій, які підвищують адаптивність системи до змін навколишнього середовища та індивідуальних потреб користувача. Одним із напрямів модернізації є впровадження додаткових сенсорів, зокрема сенсора температури, вологи чи прискорення, що дозволить, наприклад, адаптувати режими підсвітки залежно від погодних умов або активності руху.

Крім того, перспективним є впровадження більш глибокої інтеграції з хмарними сервісами та мобільними платформами. Це дозволить реалізувати централізоване керування кількома системами освітлення в межах будівлі або житлового комплексу, забезпечити логування активності, віддалене налаштування сценаріїв, а також отримання повідомлень про стан системи через мобільний застосунок або месенджери. Ще одним напрямом удосконалення може стати впровадження голосового керування або взаємодія з іншими пристроями розумного

дому через протоколи типу MQTT або Matter, що дозволить зробити систему ще зручнішою у використанні та привабливою для розгортання в комерційних та побутових умовах.

3.1.4 Вимоги до надійності системи

Система підсвітки сходів із функцією віддаленого моніторингу має відповідати високим вимогам до надійності, оскільки її призначення пов’язане з безпекою користувачів у нічний час або при недостатньому освітленні. Надійність у цьому випадку означає безперебійну роботу апаратних і програмних компонентів, своєчасне виявлення руху або присутності особи на сходах, стабільне функціонування освітлення та гарантовану відправку стану пристрою до хмарного сервера або мобільного клієнта. У разі збоїв електроживлення або втрати зв’язку система повинна бути здатною автоматично відновити свій попередній стан після перезапуску, зберігаючи поточні параметри конфігурації та освітлення.

Також важливою складовою є фізична надійність підключення: усі сенсори та елементи керування повинні бути правильно змонтовані, із дотриманням вимог до електробезпеки та електромагнітної сумісності. Надійність передачі даних між ESP32 і хмарною платформою забезпечується через стійке Wi-Fi-з’єднання, а в критичних випадках управління системою може здійснюватися локально через апаратні кнопки. Програмне забезпечення повинно передбачати логування подій, автоматичну діагностику несправностей і захист від некоректних значень, що дозволить підтримувати стабільну роботу навіть при тривалому використанні. Надійність функціонування всієї системи досягається шляхом синхронної роботи програмних потоків, використання механізмів взаємного виключення (mutex), таймінгів обробки подій та багаторівневої перевірки станів системи.

3.1.5 Вимоги до функцій та задач, які виконує система

Комп’ютерна система підсвітки сходів з можливістю дистанційного моніторингу повинна забезпечувати реалізацію ключових функцій, спрямованих на

автоматизацію освітлення та контроль його стану в режимі реального часу. Основною функцією системи є виявлення руху або присутності особи поблизу сходів, що активує підсвічування лише за потреби, тим самим підвищуючи рівень енергоефективності та комфорту користувача. Освітлення активується завдяки спрацьовуванню сенсорів (руху або присутності), дані від яких обробляються мікроконтролером ESP32. Крім того, система враховує рівень зовнішнього освітлення, що дозволяє уникнути увімкнення світла вдень або при достатній яскравості, використовуючи фотосенсори або датчики освітленості (BH1750 чи фоторезистор).

Управління всією логікою функціонування здійснюється через ESP32, який також відповідає за безпровідну передачу інформації до хмарної інфраструктури. Серед основних задач — логування стану системи, надсилання повідомлень у разі спрацювання датчиків, збереження параметрів підсвічування (колір, яскравість, режим анімації) та обробка вхідних команд з Telegram-бота чи іншого інтерфейсу. До функціоналу також входить можливість ручного управління системою через фізичні кнопки: кнопка ввімкнення/вимкнення підсвітки, кнопка перемикання режимів, кнопки налаштування кольору RGB-підсвітки. Крім цього, система повинна підтримувати кілька режимів підсвічування (статичне світло, анімації, колірна зміна) та відновлення попереднього стану після перезапуску. Усі ці функції спрямовані на створення інтелектуального освітлення, що підвищує безпеку, зручність і технологічну гнучкість для користувачів.

3.1.6 Вимоги до апаратного забезпечення

Для реалізації комп’ютерної системи підсвітки сходів з підтримкою дистанційного моніторингу апаратне забезпечення повинно відповідати вимогам стабільності, енергоефективності та функціональної гнучкості. У центрі апаратної архітектури розташовується мікроконтролер ESP32, який поєднує високі обчислювальні можливості з інтегрованими модулями Wi-Fi та Bluetooth, що дозволяє забезпечити як локальне управління системою, так і безперервну

комунікацію з хмарними сервісами. Саме на ESP32 покладається основне завдання обробки сигналів від сенсорів, формування логіки включення підсвітки, а також підтримки з'єднання з Telegram-ботом або іншими користувацькими інтерфейсами.

До мікроконтролера підключаються основні виконавчі та сигнальні компоненти системи. Для виявлення присутності людей у зоні сходів використовуються сенсори руху (наприклад, HC-SR501) або мікрохвильові сенсори (RCWL-0516), а також сенсор освітленості типу BH1750 або простий фоторезистор у комбінації з дільником напруги. В якості виконавчих пристроїв використовуються адресні світлодіодні стрічки (WS2812), які забезпечують гнучке керування кольорами та анімаціями. Апаратне управління здійснюється через кнопку живлення, кнопку зміни режимів та кнопки зміни кольору RGB. Для забезпечення збереження стану та логіки системи застосовується вбудована файлову система (SPIFFS) мікроконтролера. Таким чином, апаратне забезпечення повинно забезпечувати не лише стабільну роботу підсвітки, а й підтримку повного циклу збору, обробки, інтерпретації даних і взаємодії з віддаленим користувачем у реальному часі.

3.1.7 Вимоги до програмного забезпечення

Програмне забезпечення, що реалізує функціональну логіку системи підсвітки сходів з дистанційним моніторингом, повинно бути адаптованим до архітектури мікроконтролера ESP32, з урахуванням його апаратних обмежень і можливостей бездротової комунікації. Основна частина коду реалізується мовою програмування C++ із використанням бібліотек Arduino Core для ESP32,

Програмна логіка повинна забезпечувати обробку подій від сенсорів (руху, освітленості, присутності), керування режимами освітлення, збереження налаштувань у флеш-пам'яті та комунікацію з Telegram-ботом.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;

– графічного матеріалу:

1 Існуючі рішення організації підсвітки сходів.

2 Структурна схема комп’ютерної системи.

3 Архітектура комп’ютерної системи.

4 Мікроконтролери Arduino UNO R4 та ESP32-CAM.

5 Блок-схема алгоритму роботи комп’ютерної системи.

*Примітка: У комплект документації можуть вноситися міни та доповнення в процесі розробки.

5 Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ Етапу	Назва етапу виконання кваліфікаційної роботи	Термін виконання
1	Розробка технічного завдання	27.01 – 02.02.2025
2	Аналіз практичних рішень та вимог технічного завдання щодо системи віддаленого керування підсвіткою сходів	02.02 – 03.03.2025
3	Проєктування комп’ютерної системи підсвітки сходів з віддаленим моніторингом	04.03 – 25.04.2025
4	Програмна реалізація віддаленого моніторингу комп’ютерної системи підсвітки сходів	25.04 – 18.05.2025
5	Безпека життєдіяльності, основи охорони праці	19.05 – 28.05.2025
6	Оформлення пояснювальної записки і графічного матеріалу	28.05 – 07.06.2025
7	Перевірка на академічний плагіат, перевірка керівником та консультантами	9.06 – 15.06.2025
8	Попередній захист кваліфікаційної роботи бакалавра	16.06 – 22.06.2025
9	Захист кваліфікаційної роботи бакалавра	25.06.2025

6 Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення.

Додаток Б

Програмний код системи керування і моніторингу підсвітки сходів

```
#include <Adafruit_NeoPixel.h>
#include <SPIFFS.h>
#include <WiFiManager.h>
#include <WebSocketsClient.h>
#include <ArduinoJson.h>

//#define DEBUG

#ifdef DEBUG
  #define LOG(text)
     Serial.println(text);
#else
  #define LOG(text)
#endif

#define PIXEL_PIN 27

#define NUMPIXELS 60

#define BTN_WIFI 4
#define BTN_MODE 18
#define BTN_COLOR 19
#define BTN_POWER 21
#define BTN_R 33
#define BTN_G 32
#define BTN_B 35

#define websocketServer
     "bye1jg8i89.execute-api.eu-
     central-1.amazonaws.com"

#define getChipId()
     ((uint32_t)ESP.getEfuseMac(
     ))

#define saveDataFileName
     "/data.json"

SemaphoreHandle_t xMutexColor;
SemaphoreHandle_t xMutexStates;

struct Color {
  uint8_t R = 119;
  uint8_t G = 0;
  uint8_t B = 120;
} RGB, RGB2;

bool isChangeColor = false;

int changeColorMode = -1;

Adafruit_NeoPixel strip =
     Adafruit_NeoPixel(NUMPIXELS
     , PIXEL_PIN, NEO_GRB +
     NEO_KHZ400);

WiFiManager wm;

WebSocketsClient client;

bool toSendMessage = false;

int chatIdValue = 0;

bool isOn = false;

int currentState = -1;
int newState = 1;
int oldState = 3;

int states[] = {3, 4, 5, 6, 7,
     8};

int stateIndex = 0;

void saveData()
{
  if(xSemaphoreTake(xMutexStates
     , portMAX_DELAY) &&
     xSemaphoreTake(xMutexColor,
     portMAX_DELAY))
  {
    StaticJsonDocument<256>
     json;

    json["chatId"] =
     chatIdValue;

    json["state"] = newState;
    json["stateIndex"] =
     stateIndex;
    json["oldState"] = oldState;

    json["color"]["r"] = RGB.R;
    json["color"]["g"] = RGB.G;
    json["color"]["b"] = RGB.B;

    json["color2"]["r2"] =
     RGB2.R;
    json["color2"]["g2"] =
     RGB2.G;
    json["color2"]["b2"] =
     RGB2.B;

    json["sendMessageToServer"]
     = toSendMessage;

    File configFile =
     SPIFFS.open(saveDataFileNam
     e, "w");

    if(!configFile)
    {
```

```
      LOG("Failed to open
     file!");
    }

    //serializeJsonPretty(json,
     Serial);
    if(serializeJson(json,
     configFile) == 0)
    {
      LOG(F("Failed to write to
     file!"));
    }

    configFile.close();
    LOG("released States");
    xSemaphoreGive(xMutexStates)
     ;
    LOG("released Color");
    xSemaphoreGive(xMutexColor);
  }
}

void loadData()
{
  //SPIFFS.format();

  if(SPIFFS.begin(false) ||
     SPIFFS.begin(true))
  {
    if(SPIFFS.exists(saveDataFil
     eName))
    {
      File configFile =
     SPIFFS.open(saveDataFileNam
     e, "r");
      if(configFile)
      {
        StaticJsonDocument<256>
     json;
        DeserializationError
     error =
     deserializeJson(json,
     configFile);
        if(!error)
        {
          chatIdValue =
     json["chatId"].as<int>();


          stateIndex =
     json["stateIndex"].as<int>(
     );

          if(json["state"].as<in
     t>() == 1 &&
     json["oldState"].as<int>()
     != NULL)
          {
            oldState =
     json["oldState"].as<int>();
          }

          newState =
     json["state"].as<int>();

          RGB.R =
     json["color"]["r"].as<int>(
     );
          RGB.G =
     json["color"]["g"].as<int>(
     );
          RGB.B =
     json["color"]["b"].as<int>(
     );

          RGB2.R =
     json["color2"]["r2"].as<int
     >();
          RGB2.G =
     json["color2"]["g2"].as<int
     >();
          RGB2.B =
     json["color2"]["b2"].as<int
     >();

          toSendMessage =
     json["sendMessageToServer"]
     .as<bool>();

        }
      }
      configFile.close();
    }
  }
}

const char* getCommandName(int
     command)
{
  switch(command)
  {
    case 1:
      LOG("OFF");
      return "off";

    case 2:
      LOG("ON");
      return "on";

    case 3:
      return "for static light";

    case 4:
      return "for pong light";

    case 5:
      return "for load light";

    case 6:
      return "for random light";

    case 7:
      return "for checkers
     light";
```

```
    case 8:
      return "for two colors
     light";

    default:
      return "";
  }
}

//==============================
     ====Network functions
     START=====================
     ===========

void
     handleWebSocketMessage(cons
     t char* payload)
{
  StaticJsonDocument<256>
     jsonDoc;
  DeserializationError error =
     deserializeJson(jsonDoc,
     payload);

  if(error)
  {
    return;
  }

  String message = "";

  if(jsonDoc.containsKey("comman
     d"))
  {
    if(xSemaphoreTake(xMutexStat
     es, portMAX_DELAY))
    {
      if(jsonDoc["command"].as<i
     nt>() == 2)
      {
        newState = oldState;
      }
      else
      {
        newState =
     jsonDoc["command"].as<int>(
     );
      }

      message += "Got command "
     +
     String(getCommandName(newSt
     ate)) + ".";
      xSemaphoreGive(xMutexState
     s);
    }
  }
  if(jsonDoc.containsKey("color"
     ))
  {
    if(changeColorMode == -1)
    {

     if(xSemaphoreTake(xMutexCo
lor, portMAX_DELAY))
     {
       RGB.R =
jsonDoc["color"]["r"].as<ui
nt8_t>();
       RGB.G =
jsonDoc["color"]["g"].as<ui
nt8_t>();
       RGB.B =
jsonDoc["color"]["b"].as<ui
nt8_t>();

       if(jsonDoc.containsKey("
color2"))
       {
         RGB2.R =
jsonDoc["color2"]["r2"].as<
uint8_t>();
         RGB2.G =
jsonDoc["color2"]["g2"].as<
uint8_t>();
         RGB2.B =
jsonDoc["color2"]["b2"].as<
uint8_t>();

         message += " Changing
color 1 to " +
String(jsonDoc["color"]["r"
]) +

     ", " +
String(jsonDoc["color"]["g"
]) +

     ", " +
String(jsonDoc["color"]["b"
]) + " and color 2 to "

     +
String(jsonDoc["color2"]["r
2"]) +

     ", " +
String(jsonDoc["color2"]["g
2"]) +

     ", " +
String(jsonDoc["color2"]["b
2"]) + ".";
       }
       else
       {
         message += " Changing
color to " +
String(jsonDoc["color"]["r"
]) +

     ", " +
String(jsonDoc["color"]["g"
]) +

     ", " +
```

```
      String(jsonDoc["color"]["b"
      ]) + ".";
          }

          isChangeColor = true;

          xSemaphoreGive(xMutexCol
      or);
        }
      }
    }

    saveData();
    sendMessageToServer(message);
}

void sendMessageToServer(const
      String& message)
{
    LOG("Sending Message!");
    StaticJsonDocument<256>
      jsonDoc;

    jsonDoc["action"] =
      "sendMessage";
    jsonDoc["chat_id"] =
      chatIdValue;
    jsonDoc["device_id"] =
      String(getChipId());
    jsonDoc["message"] =
      message;

    if(xSemaphoreTake(xMutexStat
      es, portMAX_DELAY))
    {
      jsonDoc["state"] =
      newState;
      xSemaphoreGive(xMutexState
      s);
    }

    if(xSemaphoreTake(xMutexColo
      r, portMAX_DELAY))
    {
      jsonDoc["color"]["r"] =
      RGB.R;
      jsonDoc["color"]["g"] =
      RGB.G;
      jsonDoc["color"]["b"] =
      RGB.B;
      jsonDoc["color2"]["r2"] =
      RGB2.R;
      jsonDoc["color2"]["g2"] =
      RGB2.G;
      jsonDoc["color2"]["b2"] =
      RGB2.B;
      xSemaphoreGive(xMutexColor
      );
    }

    String jsonString;
    serializeJson(jsonDoc,
      jsonString);
      client.sendTXT(jsonString);
}

void initWifi()
{
  LOG("Init WiFi");

  while(digitalRead(BTN_WIFI) ==
      true)
  {
    vTaskDelay(100 /
      portTICK_RATE_MS);
  }
  char convertedValue[10];
  sprintf(convertedValue, "%d",
      chatIdValue);

  WiFiManagerParameter
      chatIdBox("chatId", "Your
      Chat Id", convertedValue,
      9);

  wm.addParameter(&chatIdBox);

  wm.setConfigPortalTimeout(120)
      ;

  if
      (!wm.startConfigPortal("Str
      ipConfig"))
    {
      LOG("failed to connect and
      hit timeout");
      wm.disconnect();
      delay(50);
    }
  else
  {
      Serial.print("Connected to
      ");
      Serial.print(wm.getWiFiSSI
      D());
      LOG();

      chatIdValue =
      atoi(chatIdBox.getValue());

      LOG("Chat id value:");
      LOG(chatIdValue);
      saveData();
  }
};

void webSocketEvent(WStype_t
      type, uint8_t * payload,
      size_t length)
{
  if(!WiFi.isConnected())
  {
    return;
  }
  switch (type) {
    case WStype_DISCONNECTED:
```

```
      LOG("Disconnected");
      connectToWebSocket();
      break;
    case WStype_CONNECTED:
      LOG("Connected");
      if(toSendMessage)
      {
        sendMessageToServer("Dev
     ice now connected to server
     :). Uploading local
     changes.... ");
        toSendMessage = false;
      }
      break;
    case WStype_TEXT:
      LOG("Received message: ");
      LOG((char*)payload);
      handleWebSocketMessage((co
     nst char*)payload);
      break;
    default:
      break;
  }
}

void connectToWebSocket()
{
  String url;
  if(xSemaphoreTake(xMutexStates
     , portMAX_DELAY))
    {
      if(xSemaphoreTake(xMutexCo
     lor, portMAX_DELAY))
      {

      url =
     String("/production/?chat_i
     d=") +
                String(chatIdVal
     ue) + String("&device_id=")
     +
                String(getChipId
     ()) + String("&state=") +
                String(currentSt
     ate) +
                "&r=" +
     String(RGB.R) + "&g=" +
     String(RGB.G) + "&b=" +
     String(RGB.B) +
                "&r2=" +
     String(RGB2.R) + "&g2=" +
     String(RGB2.G) + "&b2=" +
     String(RGB2.B);

      xSemaphoreGive(xMutexColor
     );
    }
    xSemaphoreGive(xMutexStates)
     ;
  }

  client.beginSSL(websocketServe
     r, 443, url, "", "wss");
  client.onEvent(webSocketEvent)
     ;
}

void tryToConnectToWiFi()
{
  if(!wm.getWiFiIsSaved())
  {
    return;
  }

  auto SSID = wm.getWiFiSSID();

  int networks =
     WiFi.scanNetworks();

  for(size_t i = 0; i <
     networks; i++)
  {
    String foundSSID =
     WiFi.SSID(i);
    if(foundSSID == SSID)
    {
      wm.setEnableConfigPortal(f
     alse);
      wm.setDebugOutput(true);
      wm.setConfigPortalTimeout(
     30);
      wm.autoConnect();
      return;
    }
  }
}

void taskNetwork(void*
     parameters)
{
  LOG("Task Netwok.");
  for(;;){
    if(WiFi.isConnected())
    {
      client.loop();
    }

    if(wm.getWiFiIsSaved() &&
     !WiFi.isConnected())
    {
      tryToConnectToWiFi();
    }

    if(digitalRead(BTN_WIFI) ==
     HIGH)
    {
      LOG("Pressed.");
      initWifi();
    }

    vTaskDelay(50 /
     portTICK_RATE_MS);
  }
}
```

```
//==============================
    ====Network functions
    END=======================
    =========

//==============================
    ====Strip functions
    START=====================
    ===========

void turnOnOff()
{
  while(digitalRead(BTN_POWER)
    == HIGH)
  {
    vTaskDelay(200 /
    portTICK_RATE_MS);
  }
  if(xSemaphoreTake(xMutexStates
    , portMAX_DELAY))
    {
      LOG("On/Off");

      isOn = !isOn;

      vTaskDelay(200 /
    portTICK_RATE_MS);

      currentState = -1;

      if(newState >= 2)
      {
        oldState = newState;
      }

      if(!isOn)
      {
        newState = 1;
        strip.clear();
        strip.fill(strip.Color(0
    , 0, 0), 0, NUMPIXELS);
        strip.show();
        if(WiFi.isConnected())
        {
          String message =
    "Device is off";
          sendMessageToServer(me
    ssage);
        }
      }
      else
      {
        if(WiFi.isConnected())
        {
          String message =
    "Device is on";
          sendMessageToServer(me
    ssage);
        }
        newState = oldState;
      }
      xSemaphoreGive(xMutexState
    s);
    }

    saveData(); //Remove if
    unexpected behaviour
}

bool checkForChanges()
{
  if (changeColorMode >= 1) {
    return true;
  }

  if(xSemaphoreTake(xMutexStates
    , 0) &&
    xSemaphoreTake(xMutexColor,
    0))
    {
      if(currentState !=
    newState || isChangeColor
    || changeColorMode >= 1)
        {
          currentState = -1;
          isChangeColor = false;
          strip.clear();
          strip.fill(strip.Color
    (0, 0, 0), 0, NUMPIXELS);
          strip.show();
          xSemaphoreGive(xMutexS
    tates);
          xSemaphoreGive(xMutexC
    olor);
          return true;
        }
      xSemaphoreGive(xMutexState
    s);
      xSemaphoreGive(xMutexColor
    );
      return false;
    }

    return false;
}

void staticLight()
{
  LOG("1");
  strip.clear();
  if(xSemaphoreTake(xMutexColor,
    0))
  {
    strip.fill(strip.Color(RGB.R
    , RGB.G, RGB.B), 0,
    NUMPIXELS);
    xSemaphoreGive(xMutexColor);
  }
  else
  {
    LOG("OUT Color");
    return;
  }
  strip.show();
```

```
  if(xSemaphoreTake(xMutexStates
     , 0))
  {
    currentState = newState;
    xSemaphoreGive(xMutexStates)
     ;
  }
  else
  {
    LOG("OUT States")
    return;
  }
  if(WiFi.isConnected())
  {
    String message = "Device
     glows statically!";
    sendMessageToServer(message)
     ;
  }
  else
  {
    toSendMessage = true;
  }
}

void pongLight()
{
  strip.clear();

  LOG("2");

  if(xSemaphoreTake(xMutexStates
     , 0))
  {
    currentState = newState;
    xSemaphoreGive(xMutexStates)
     ;
  }
  else
  {
    LOG("OUT States");
    return;
  }

  uint8_t r = 0;
  uint8_t g = 0;
  uint8_t b = 0;

  if(xSemaphoreTake(xMutexColor,
     0))
  {
    r = RGB.R;
    g = RGB.G;
    b = RGB.B;

    xSemaphoreGive(xMutexColor);
  }
  else
  {
    currentState = -1;
    LOG("OUT Color");
    return;
  }

if(WiFi.isConnected())
{
  String message = "Device
   running pong style light!";
  sendMessageToServer(message)
   ;
}
else
{
  toSendMessage = true;
}


while(true)
{
  int pixelN = 0;

  while(pixelN != NUMPIXELS)
  {
    vTaskDelay(50 /
   portTICK_RATE_MS);

    if(pixelN == 0)
    {
      strip.setPixelColor(pixe
   lN, r, g, b);
      pixelN++;
      strip.show();
      continue;
    }

    strip.setPixelColor(pixelN
   - 1, 0, 0, 0);
    strip.setPixelColor(pixelN
   , r, g, b);
    strip.show();
    if(pixelN % 5 == 0)
    {
      if(checkForChanges())
      {
        return;
      }
    }
    pixelN++;
  }

  while(pixelN != -1)
  {
    vTaskDelay(50 /
   portTICK_RATE_MS);

    if(pixelN == NUMPIXELS -
   1)
    {
      strip.setPixelColor(pixe
   lN, r, g, b);
      pixelN--;
      strip.show();
      continue;
    }
```

```
      strip.setPixelColor(pixelN
     + 1, 0, 0, 0);
      strip.setPixelColor(pixelN
     , r, g, b);
      strip.show();

      if(pixelN % 5 == 0)
      {
        if(checkForChanges())
        {
          return;
        }
      }

      pixelN--;
    }
  }
}

void loadLight()
{
  strip.clear();

  LOG("5");

  if(xSemaphoreTake(xMutexStates
     , 0))
  {
    currentState = newState;
    xSemaphoreGive(xMutexStates)
     ;
  }
  else
  {
    LOG("OUT States");
    return;
  }

  uint8_t r = 0;
  uint8_t g = 0;
  uint8_t b = 0;

  if(xSemaphoreTake(xMutexColor,
     0))
  {
    r = RGB.R;
    g = RGB.G;
    b = RGB.B;

    xSemaphoreGive(xMutexColor);
  }
  else
  {
    currentState = -1;
    LOG("OUT Color");
    return;
  }

  if(WiFi.isConnected())
  {
    String message = "Device
     running load style light!";
    sendMessageToServer(message)
     ;
  }
  else
  {
    toSendMessage = true;
  }

  while(true)
  {
    int pixelN = 0;

    while(pixelN != NUMPIXELS)
    {
      vTaskDelay(50 /
     portTICK_RATE_MS);

      strip.setPixelColor(pixelN
     , r, g, b);
      strip.show();

      if(pixelN % 5 == 0)
      {
        if(checkForChanges())
        {
          return;
        }
      }

      pixelN++;
    }

    while(pixelN != -1)
    {
      vTaskDelay(50 /
     portTICK_RATE_MS);

      strip.setPixelColor(pixelN
     , 0, 0, 0);
      strip.show();

      if(pixelN % 5 == 0)
      {
        if(checkForChanges())
        {
          return;
        }
      }

      pixelN--;
    }
  }
}

void randomLight()
{
  strip.clear();

  LOG("6");

  if(xSemaphoreTake(xMutexStates
     , 0))
  {
```

```
  currentState = newState;
  xSemaphoreGive(xMutexStates)
   ;
}
else
{
  LOG("OUT States");
  return;
}

uint8_t r = 0;
uint8_t g = 0;
uint8_t b = 0;

if(xSemaphoreTake(xMutexColor,
   0))
{
  r = RGB.R;
  g = RGB.G;
  b = RGB.B;

  xSemaphoreGive(xMutexColor);
}
else
{
  currentState = -1;
  LOG("OUT Color");
  return;
}

if(WiFi.isConnected())
{
  String message = "Device
   running random style
   light!";
  sendMessageToServer(message)
   ;
}
else
{
  toSendMessage = true;
}

int counter = 0;

while(true)
{
    vTaskDelay(100 /
   portTICK_RATE_MS);

    int pixelN = random(0,
   NUMPIXELS);

    strip.setPixelColor(pixelN
   , r, g, b);
    strip.show();

    strip.setPixelColor(pixelN
   , 0, 0, 0);

    counter = (counter + 1) %
   (NUMPIXELS - 1);

    if(pixelN % 5 == 0)
    {
      if(checkForChanges())
      {
        return;
      }
    }
  }
}

void checkersLight()
{
  strip.clear();

  LOG("7");

  if(xSemaphoreTake(xMutexStates
     , 0))
  {
    currentState = newState;
    xSemaphoreGive(xMutexStates)
     ;
  }
  else
  {
    LOG("OUT States");
    return;
  }

  uint8_t r = 0;
  uint8_t g = 0;
  uint8_t b = 0;

  uint8_t r2 = 0;
  uint8_t g2 = 0;
  uint8_t b2 = 0;

  if(xSemaphoreTake(xMutexColor,
     0))
  {
    r = RGB.R;
    g = RGB.G;
    b = RGB.B;

    r2 = RGB2.R;
    g2 = RGB2.G;
    b2 = RGB2.B;

    xSemaphoreGive(xMutexColor);
  }
  else
  {
    currentState = -1;
    LOG("OUT Color");
    return;
  }

  if(WiFi.isConnected())
  {
    String message = "Device
     running checkers style
     light!";
```

```
    sendMessageToServer(message)
     ;
  }
  else
  {
    toSendMessage = true;
  }

  int stage = 0;

  while(true)
  {
    for(size_t i = 0; i <
     NUMPIXELS; i += 2)
    {
      if(stage == 0)
        strip.setPixelColor(i,
     r, g, b);
      else
        strip.setPixelColor(i,
     r2, g2, b2);
    }

    for(size_t i = 1; i <
     NUMPIXELS; i += 2)
    {
      if(stage == 1)
        strip.setPixelColor(i,
     r, g, b);
      else
        strip.setPixelColor(i,
     r2, g2, b2);
    }

    strip.show();

    if(checkForChanges())
    {
      return;
    }

    vTaskDelay(3000 /
     portTICK_RATE_MS);

    if(checkForChanges())
    {
      return;
    }

    if(++stage == 2)
      stage = 0;
  }
}

void twoColorsLight()
{
  strip.clear();

  LOG("8");

  if(xSemaphoreTake(xMutexStates
     , 0))
  {
    currentState = newState;
    xSemaphoreGive(xMutexStates)
     ;
  }
  else
  {
    LOG("OUT States");
    return;
  }

  uint8_t r = 0;
  uint8_t g = 0;
  uint8_t b = 0;

  uint8_t r2 = 0;
  uint8_t g2 = 0;
  uint8_t b2 = 0;

  if(xSemaphoreTake(xMutexColor,
     0))
  {
    r = RGB.R;
    g = RGB.G;
    b = RGB.B;

    r2 = RGB2.R;
    g2 = RGB2.G;
    b2 = RGB2.B;

    xSemaphoreGive(xMutexColor);
  }
  else
  {
    currentState = -1;
    LOG("OUT Color");
    return;
  }

  if(WiFi.isConnected())
  {
    String message = "Device
     running two colors style
     light!";
    sendMessageToServer(message)
     ;
  }
  else
  {
    toSendMessage = true;
  }

  int stage = 0;

  while(true)
  {
    for(size_t i = 0; i <
     NUMPIXELS; i++)
    {
      if(stage == 0)
        strip.setPixelColor(i,
     r, g, b);
      else
```

```
        strip.setPixelColor(i,
     r2, g2, b2);

      if(i % 10 == 0)
      {
        if(checkForChanges())
        {
          return;
        }
      }
    }

    for(size_t i = 0; i <
     NUMPIXELS; i++)
    {
      if(stage == 1)
        strip.setPixelColor(i,
     r, g, b);
      else
        strip.setPixelColor(i,
     r2, g2, b2);

      if(i % 10 == 0)
      {
        if(checkForChanges())
        {
          return;
        }
      }
    }
    strip.show();

    vTaskDelay(7000 /
     portTICK_RATE_MS);

    if(++stage == 2)
      stage = 0;

  }
}

void changeMode(int index)
{
  while(digitalRead(BTN_MODE) ==
     HIGH)
  {
    vTaskDelay(200 /
     portTICK_RATE_MS);
  }

  if(xSemaphoreTake(xMutexStates
     , portMAX_DELAY))
  {
    newState = states[index];
    xSemaphoreGive(xMutexStates)
     ;
  }

  stateIndex = (stateIndex + 1)
     % (sizeof(states) /
     sizeof(states[0]));

  saveData();
}

void changeColor()
{
  LOG("Changing Color!!!!");

  while(digitalRead(BTN_COLOR)
     == HIGH)
  {
    vTaskDelay(500 /
     portMAX_DELAY);
  }

  if(xSemaphoreTake(xMutexColor,
     portMAX_DELAY))
  {
  strip.clear();
  strip.fill(strip.Color(RGB.R,
     RGB.G, RGB.B), 0,
     NUMPIXELS);
  strip.show();

    while(changeColorMode == 1)
    {
      if(digitalRead(BTN_COLOR)
     == HIGH)
      {
        while(digitalRead(BTN_CO
     LOR) == HIGH)
        {
          vTaskDelay(500 /
     portMAX_DELAY);
        }
        LOG("Changing Color
     out!");
        changeColorMode = 2;

        strip.clear();
        strip.fill(strip.Color(R
     GB2.R, RGB2.G, RGB2.B), 0,
     NUMPIXELS);
        strip.show();
      }

      if(digitalRead(BTN_R) ==
     HIGH)
      {
        LOG("R Pressed");
        RGB.R = (uint8_t)((RGB.R
     + 5) % 256);
        strip.fill(strip.Color(R
     GB.R, RGB.G, RGB.B), 0,
     NUMPIXELS);
        strip.show();
        vTaskDelay(pdMS_TO_TICKS
     (400));
      }

      if(digitalRead(BTN_G) ==
     HIGH)
      {
        LOG("g Pressed");
```

```
    RGB.G = (uint8_t)((RGB.G
+ 5) % 256);
    strip.fill(strip.Color(R
GB.R, RGB.G, RGB.B), 0,
NUMPIXELS);
    strip.show();
    vTaskDelay(pdMS_TO_TICKS
(400));
  }

  if(digitalRead(BTN_B) ==
HIGH)
  {
    LOG("B Pressed");
    RGB.B = (uint8_t)((RGB.B
+ 5) % 256);
    strip.fill(strip.Color(R
GB.R, RGB.G, RGB.B), 0,
NUMPIXELS);
    strip.show();
    vTaskDelay(pdMS_TO_TICKS
(400));
  }

}

while(changeColorMode == 2)
{
  if(digitalRead(BTN_COLOR)
== HIGH)
  {
    while(digitalRead(BTN_CO
LOR) == HIGH)
    {
      vTaskDelay(500 /
portMAX_DELAY);
    }
    LOG("Changing Color
out!");
    changeColorMode = -1;
    newState = oldState;
    currentState = -1;
  }

  if(digitalRead(BTN_R) ==
HIGH)
  {
    LOG("R Pressed");
    RGB2.R =
(uint8_t)((RGB2.R + 5) %
256);
    strip.fill(strip.Color(R
GB2.R, RGB2.G, RGB2.B), 0,
NUMPIXELS);
    strip.show();
    vTaskDelay(pdMS_TO_TICKS
(400));
  }

  if(digitalRead(BTN_G) ==
HIGH)
  {
    LOG("g Pressed");
    RGB2.G =
(uint8_t)((RGB2.G + 5) %
256);
    strip.fill(strip.Color(R
GB2.R, RGB2.G, RGB2.B), 0,
NUMPIXELS);
    strip.show();
    vTaskDelay(pdMS_TO_TICKS
(400));
  }

  if(digitalRead(BTN_B) ==
HIGH)
  {
    LOG("B Pressed");
    RGB2.B =
(uint8_t)((RGB2.B + 5) %
256);
    strip.fill(strip.Color(R
GB2.R, RGB2.G, RGB2.B), 0,
NUMPIXELS);
    strip.show();
    vTaskDelay(pdMS_TO_TICKS
(400));
  }
}

  isChangeColor = true;
  strip.clear();
  strip.show();
  xSemaphoreGive(xMutexColor);
  LOG("Released Color");
}

  saveData();
}

void taskStrip (void*
    parameters)
{
  LOG("Task Strip.");
  for(;;)
  {
    if(!xSemaphoreTake(xMutexSta
    tes, 0))
    {
      continue;
    }
    else
    {
      xSemaphoreGive(xMutexState
    s);
    }

    if((newState >= 2 && !isOn)
    || (newState == 1 && isOn))
    {
      turnOnOff();
      vTaskDelay(200 /
    portTICK_RATE_MS);
    }

    if(changeColorMode >= 1)
```

```
    {
      vTaskDelay(200 /
     portTICK_RATE_MS);
      continue;
    }

    if(isOn && (currentState !=
     newState))
    {
      switch(newState)
      {
        case 3:
          LOG("Static Light.");
          staticLight();
          break;
        case 4:
          LOG("Pong Light.");
          pongLight();
          break;
        case 5:
          LOG("Load Loght.");
          loadLight();
          break;
        case 6:
          LOG("Random Light.");
          randomLight();
          break;
        case 7:
          LOG("Checker Light.");
          checkersLight();
          break;
        case 8:
          LOG("Two Colors
     Light");
          twoColorsLight();
          break;
        default:
          break;
      }
    }

    vTaskDelay(200 /
     portTICK_RATE_MS);
  }
}

void taskButtons(void*
     parameters)
{
  LOG("Task buttons.");
  for(;;)
  {

    if(digitalRead(BTN_POWER) ==
     HIGH)
    {
      LOG("Power button
     pressed.");
      turnOnOff();
      vTaskDelay(200 /
     portTICK_RATE_MS);
      changeColorMode = -1;
    }

    if(changeColorMode >= 1 ||
     !isOn)
    {
      vTaskDelay(200 /
     portTICK_RATE_MS);
      continue;
    }

    if(digitalRead(BTN_MODE) ==
     HIGH)
    {
      LOG("Mode button
     pressed.");
      changeMode(stateIndex);
      vTaskDelay(200 /
     portTICK_RATE_MS);
    }

    if(digitalRead(BTN_COLOR) ==
     HIGH)
    {
      LOG("Color button
     pressed.");
      if(xSemaphoreTake(xMutexSt
     ates, portMAX_DELAY))
      {
        oldState = newState;
        changeColorMode = 1;
        newState = 2;
        xSemaphoreGive(xMutexSta
     tes);
      }
      vTaskDelay(200 /
     portTICK_RATE_MS);
      changeColor();
    }

    vTaskDelay(200 /
     portTICK_RATE_MS);
  }
}

//==============================
     ====Strip functions
     END=======================
     =========

void setup()
{
  Serial.begin(9600);

  strip.begin();
  strip.setBrightness(150);

  loadData();

  delay(500);

  //wm.resetSettings();
```

```
wm.setEnableConfigPortal(false
   );
wm.setDebugOutput(false);

xMutexColor =
   xSemaphoreCreateMutex();
xMutexStates =
   xSemaphoreCreateMutex();

if (xMutexColor == NULL) {
  LOG("Failed to create
   xMutexColor!");
  ESP.restart();
}

if (xMutexStates == NULL) {
  LOG("Failed to create
   xMutexStates!");
  ESP.restart();
}

String url = "/production";
client.beginSSL(
    websocketServer,
    443,
    url,
    "",
    "wss");
client.onEvent(webSocketEvent)
   ;

if (wm.getWiFiIsSaved()) {
  bool res;
  wm.setConfigPortalTimeout(30
   );
  res = wm.autoConnect();

  if(!res)
  {
    wm.disconnect();
  }
}

pinMode(BTN_WIFI, INPUT);
pinMode(BTN_POWER, INPUT);
pinMode(BTN_COLOR, INPUT);
pinMode(BTN_MODE, INPUT);
pinMode(BTN_R, INPUT);
pinMode(BTN_G, INPUT);
pinMode(BTN_B, INPUT);
pinMode(PIXEL_PIN, OUTPUT);

BaseType_t taskResult;

taskResult =
   xTaskCreatePinnedToCore(
  taskNetwork,
  "Network Task",
  32768,
  NULL,
  1,
  NULL,
  1
  );

  delay(500);

  if(taskResult != pdPASS)
  {
    LOG("Task Failed 1");
    ESP.restart();
  }

  taskResult =
     xTaskCreatePinnedToCore(
    taskStrip,
    "Strip Task",
    32768,
    NULL,
    1,
    NULL,
    1
  );

  delay(500);

  if(taskResult != pdPASS)
  {
    LOG("Task Failed 2");
    ESP.restart();
  }

  taskResult =
     xTaskCreatePinnedToCore(
    taskButtons,
    "Task for buttons",
    32768,
    NULL,
    1,
    NULL,
    1
  );

  delay(500);

  if(taskResult != pdPASS)
  {
    LOG("Task Failed 3");
    ESP.restart();
  }

  delay(500);
}

void loop()
{}
```