Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

Бакалаврська кваліфікаційна робота
на тему:
«Розробка комунікаційного інтерфейсу для мікроконтролера Arduino за технологією Bluetooth»

Виконав: студент 4 курсу, групи 2АКІТ-21б
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології

Денис ІЩЕПЛЕНСЬКИЙ

Керівник: д.т.н., проф., професор каф. КСУ

В'ячеслав КОВТУН

«10» червня 2025 р.

Рецензент: к.т.н., доцент каф. АІІТ

Юрій ІВАНОВ

«11» червня 2025 р.

Допущено до захисту
Зав. кафедри КСУ

В'ячеслав Ковтун
«12» червня 2025р.

Вінниця ВНТУ – 2025 рік

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 15 – Автоматизація та приладобудування
Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо - професійна програма – Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
Д.т.н., проф.
_____В'ячеслав КОВТУН
"24" 05 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Щепленському Денису Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка комунікаційного інтерфейсу для мікроконтролера Arduino за технологією Bluetooth
керівник роботи Ковтун В'ячеслав Васильович
затверджені наказом ВНТУ № 97 від 20.03.2025 р.
2. Термін подання студентом роботи 10 червня 2025 року
3. Вихідні дані до роботи:

1. Collotta M., Pau G. Bluetooth for Internet of Things: A fuzzy approach to improve power management in smart homes. *Computers & Electrical Engineering*. 2015. Vol. 44. P. 137–152.
2. D. Dragomir, L. Gheorghe, S. Costea, A. Radovici A Survey on Secure Communication Protocols for IoT Systems. 2016.
3. Ferreira E., Páris C., Roseiro L. Web API BLE communication using. Net Core. IEEE, 2022. ISBN 989-33-3436-5.
4. Gast M. S. Building Applications with iBeacon: Proximity and Location Services with Bluetooth Low Energy. O'Reilly Media, 2014. ISBN 978-1-4919-0483-1.

4. Зміст текстової частини:
Анотація; Зміст; Вступ; .
Розділ 1. Аналіз предметної області комунікаційних інтерфейсів.
Розділ 2. Проєктування системи.
Розділ 3. Реалізація.
Розділ 4. Тестування та валідація.
Висновки, список використаних джерел, додатки

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): Таксономія BLE-додатків, їхні проблеми та рішення;
Електрична схема підключення; Загальний вигляд користувацького інтерфейсу;

Блок-схема алгоритму користувацького досвіду; Вебінтерфейс системи; Пошук пристроїв для з'єднання; Вибір пристрою для з'єднання; Елементи керування освітленням; Результати тестування статусної строки стану системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	Виконая прийняв

7. Дата видачі завдання 26.05.2025 р.

КАЛЕНДАРНИЙ ПЛАН

1	Вступ: актуальність, мета, завдання, новизна, практична цінність.	17 травня 2025 р.
2	Аналіз об'єкта автоматизації і техніко-економічне обґрунтування доцільності розробки.	20 травня 2025 р.
3	Постановка задачі і розробка технічного завдання	26 травня 2025 р.
4	Аналіз і вибір технічних і програмних засобів	30 травня 2025 р.
5	Вибір та розрахунок технічних засобів автоматизації.	03 червня 2025 р.
6	Розробка програмного забезпечення системи	08 червня 2025 р.
7	Опис системи автоматизації та алгоритм роботи.	10 червня 2025 р.
8	Оформлення пояснювальної записки	12 червня 2025 р.
9	Графічні матеріали:	
	Таксономія BLE-додатків, їхні проблеми та рішення	02 червня 2025 р.
	Електрична схема підключення	02 червня 2025 р.
	Загальний вигляд користувацького інтерфейсу	03 червня 2025 р.
	Блок-схема алгоритму користувацького досвіду	03 червня 2025 р.
	Вебінтерфейс системи	03 червня 2025 р.
	Пошук пристроїв для з'єднання	04 червня 2025 р.
	Вибір пристрою для з'єднання	04 червня 2025 р.
	Елементи керування освітленням	06 червня 2025 р.
	Результати тестування статусної строки стану системи	06 червня 2025 р.
10	Захист БДР	18 червня. 2025 р.

Студент _____ Денис ЩЕПЛЕНСЬКИЙ
(підпис)

Керівник роботи _____ В'ячеслав КОВТУН
(підпис)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 52 сторінок формату А4, на яких розміщено 9 рисунків, а список використаних джерел містить 25 найменувань.

Метою роботи є створення та апробація ефективного інтерфейсу керування LED-освітленням через веббраузер із використанням технології Bluetooth Low Energy (BLE). Це, своєю чергою, передбачає автоматичний розрахунок подій на основі даних про сонячний цикл.

У роботі проведено аналіз протоколу BLE, платформ Arduino та Web Bluetooth API. Обґрунтовано вибір ESP32 DevKitC та вебтехнологій для клієнтського рішення. Сформульовано функціональні та нефункціональні вимоги, спроєктовано апаратну архітектуру на базі ESP32 та MOSFET-драйверів. Розроблено BLE-GATT-сервіси для керування освітленням та діагностики. Реалізовано клієнтський вебдодаток із застосуванням HTML, CSS та JavaScript для ініціалізації з’єднання, керування параметрами та автоматизації за сонячним циклом. Проведено комплексне тестування, що підтвердило відповідність системи технічним вимогам та зручність використання. Виявлено напрями подальшого вдосконалення.

Ключові слова: інтерфейс, мікроконтролер Arduino, технологія Bluetooth, BLE-GATT-сервіси

ABSTRACT

The bachelor's thesis consists of 52 A4 pages, with 9 figures, and a list of references containing 25 titles.

The aim of the work is to create and test an effective interface for controlling LED lighting through a web browser using Bluetooth Low Energy (BLE) technology. This, in turn, involves the automatic calculation of events based on solar cycle data.

The paper analyzes the BLE protocol, Arduino platforms, and the Web Bluetooth API. The choice of ESP32 DevKitC and web technologies for the client solution is substantiated. Functional and non-functional requirements are formulated, and a hardware architecture based on ESP32 and MOSFET drivers is designed. BLE-GATT services for lighting control and diagnostics were developed. A client web application was implemented using HTML, CSS, and JavaScript to initialize the connection, manage parameters, and automate according to the solar cycle. Comprehensive testing was conducted, which confirmed the system's compliance with technical requirements and ease of use. Areas for further improvement were identified.

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Комунікаційних інтерфейсів 6
1.1. Технологія Bluetooth Low Energy (BLE) 6
1.2. Платформи Arduino з підтримкою BLE 10
1.3. Web Bluetooth API: можливості та обмеження 12
1.4. Огляд існуючих комунікаційних інтерфейсів 16
1.5. Висновки за розділом 17
РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ 19
2.1. Постановка технічного завдання 19
2.2. Функціональні та нефункціональні вимоги 20
2.3. Архітектура апаратного рівня 21
2.4. Проєктування BLE-GATT-сервісів і характеристик 26
2.5. Проєктування користувацького інтерфейсу вебдодатка та алгоритму користування вебсистемою 30
2.6. Висновки за розділом 35
РОЗДІЛ 3 РЕАЛІЗАЦІЯ 36
3.1. Розробка прошивки Arduino 36
3.2. Реалізація BLE-GATT-логіки та ефектів 37
3.3. Створення вебінтерфейсу (HTML, CSS, JavaScript) 38
3.4. Інтеграція фронтенду й прошивки через Web Bluetooth API 40
3.5. Обробка геолокації та автоматизація за сонячним циклом 42
3.6. Висновки за розділом 43
РОЗДІЛ 4 ТЕСТУВАННЯ ТА ВАЛІДАЦІЯ 44
4.1. Методика та сценарії тестування 44
4.2. Функціональні випробування 45
4.3. Тест стійкості з'єднання та перепідключення 49
4.4. Оцінка юзабіліті вебінтерфейсу 50
4.5. Аналіз результатів 51
4.6. Висновки за розділом 52
ВИСНОВКИ 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55
Додаток А (обов’язковий) Протокол перевірки на плагіат 59
Додаток Б (обов’язковий) Технічне завдання 60
Додаток В (довідковий) Лістинг Програми 65
Додаток Г (обов'язковий) Ілюстративна частина 90

ВСТУП

Обґрунтування вибору теми дослідження. В умовах динамічного розвитку технологій «розумного дому» та зростання попиту на інтуїтивні інтерфейси керування побутовими приладами набуває особливої ваги дослідження безпосередньої інтеграції веббраузера з мікроконтролером через Bluetooth Low Energy (BLE). Загальновідомо, що більшість сучасних систем домашньої автоматизації працюють за архітектурою «клієнт–сервер», що створює додаткові точки відмови та збільшує затримки при обміні даними між користувачем і пристроєм. Водночас основними проблемами існуючих рішень є залежність від стабільності мережі Wi-Fi, підвищене навантаження на сервери та обмежена автономність у випадку відключення інтернету. Однією з перспективних методик вирішення цих недоліків є застосування Web Bluetooth API, яке забезпечує прямий зв’язок між браузером на мобільному пристрої та мікроконтролером ESP32 без проміжного серверного компоненту.

У рамках даного дослідження було розроблено клієнтський вебдодаток для керування двоканальною LED-стрічкою (теплий та холодний білий канал) з автоматизацією процесу вмикання/вимикання за часом сходу і заходу сонця на основі геолокації користувача.

Метою дослідження є створення та апробація ефективного інтерфейсу керування LED-освітленням через веббраузер з використанням BLE з автоматичним розрахунком подій на основі даних про сонячний цикл.

Для досягнення поставленої мети сформульовано такі завдання:

1) здійснити аналіз протоколу BLE та можливостей Web Bluetooth API;
2) провести огляд існуючих комунікаційних інтерфейсів для BLE-пристроїв;
3) спроєктувати апаратну та програмну архітектуру системи керування;
4) розробити вебдодаток з реалізацією GATT-сервісів для керування яскравістю, температурою та режимами освітлення;
5) реалізувати механізм отримання геолокації та запитів до API часу сходу/заходу сонця;

6) провести тестування та оцінку продуктивності передачі даних і стабільності роботи інтерфейсу.

Об’єктом дослідження є процес автоматизованого керування побутовим освітленням у домашніх системах.

Предмет дослідження становлять методи реалізації Web Bluetooth API та алгоритми автоматизації подій на основі геолокації та даних про сонячний цикл.

Новизна полягає у створенні власної технічної реалізації клієнтського рішення для керування BLE-пристроєм із автоматичним розрахунком часу включення та виключення з використанням даних прогнозу погоди.

Практична цінність одержаних результатів забезпечується можливістю швидкого розгортання готового вебдодатка для управління побутовим освітленням.

Результати роботи апробовано у лабораторних умовах Вінницького національного технічного університету.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ КОМУНІКАЦІЙНИХ ІНТЕРФЕЙСІВ

1.1. Технологія Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) є енергоефективним бездротовим протоколом, розробленим для передачі невеликих обсягів даних із низьким енергоспоживанням. Оскільки традиційний Bluetooth Classic споживає значно більше ресурсів, BLE отримав широке поширення в пристроях Інтернету речей (IoT), медичній апаратурі та системах домашньої автоматизації[1, 4, 21, 23]. На рис. 1.1 наведено таксономію BLE-додатків, їхні проблеми та рішення.

Рисунок 1.1 – Таксономія BLE-додатків, їхні проблеми та рішення (перекладено зі статті [24])

Таким чином, BLE-GATT архітектура базується на моделі «сервер — клієнт», де сервер надає набір сервісів і характеристик, а клієнт виконує операції читання, запису або підписки на оновлення значень. У контексті даного

дослідження саме реалізація GATT-сервісів на ESP32 дозволяє гнучко керувати параметрами освітлення.

Зокрема, характеристики BLE-GATT:

- Service UUID - унікальний ідентифікатор сервісу, що слугує для розрізнення різних функціональних блоків у структурі BLE-пристрою. Завдяки цьому ідентифікатору клієнтський пристрій має змогу ефективно знаходити та взаємодіяти з конкретними сервісами, які надаються BLE-сервером. Це, своєю чергою, дозволяє забезпечити організовану та стандартизовану взаємодію між пристроями, що є критично важливим для масштабованості систем Інтернету речей.

При проєктуванні системи керування освітленням було визначено, що кожному функціональному блоку, такому як керування освітленням або діагностика, слід присвоїти унікальний Service UUID. Такий підхід забезпечує чітке розмежування між різними типами даних та операцій, що спрощує процес розробки клієнтського застосунку та підвищує надійність обміну даними. Таким чином, використання Service UUID гарантує однозначну ідентифікацію сервісів та спрощує їх інтеграцію в загальну архітектуру системи.

Отже, Service UUID відіграє ключову роль у створенні модульної та розширюваної архітектури BLE-систем. Він є основою для ефективного пошуку та підключення до відповідних функціональних блоків на сервері, що, у свою чергу, оптимізує взаємодію між пристроями та знижує складність розробки програмного забезпечення. Це особливо актуально в умовах, коли система передбачає підтримку кількох різних функцій або потребує інтеграції з різноманітними клієнтськими пристроями.

- Characteristic UUID - унікальний ідентифікатор параметра, який дає змогу точно визначити конкретне значення або атрибут у межах певного сервісу BLE. Цей ідентифікатор є необхідним для того, щоб клієнт міг однозначно звертатися до таких параметрів, як яскравість, температура або режим роботи освітлення, забезпечуючи контрольований та структурований обмін даними. Завдяки

використанню Characteristic UUID досягається високий рівень деталізації в управлінні пристроєм, що, у свою чергу, сприяє гнучкості системи.

У контексті розробки комунікаційного інтерфейсу для мікроконтролера Arduino Characteristic UUID було застосовано для ідентифікації параметрів, що відповідають за керування LED-освітленням. Зокрема, для регулювання яскравості, встановлення колірної температури та вибору режимів освітлення було визначено окремі Characteristic UUID. Такий підхід дозволив розмежувати потоки даних та забезпечити їх коректну інтерпретацію на боці як клієнта, так і сервера, що є важливим для точності та ефективності керування.

Таким чином, Characteristic UUID є фундаментальним елементом архітектури BLE GATT, що забезпечує адресність та специфічність доступу до даних пристрою. Його застосування дозволяє створювати чітко визначені інтерфейси для взаємодії, що спрощує інтеграцію нових функцій та підтримує стандартизацію протоколів обміну. Отже, це сприяє підвищенню надійності та розширюваності розробленої системи.

- Properties (Read, Write, Notify) - визначають можливості операцій з характеристикою BLE, встановлюючи її поведінкові атрибути та можливості взаємодії. Ці властивості є ключовими для визначення функціональності кожної характеристики, оскільки вони регламентують, чи може клієнт лише читати значення (Read), записувати його (Write), чи отримувати сповіщення про його зміни (Notify). Завдяки цим властивостям забезпечується керованість доступу до даних та ефективність комунікації між пристроями.

У розробленій системі кожній характеристиці було присвоєно відповідні властивості для забезпечення необхідної функціональності. Наприклад, для керування яскравістю та температурою використовувалася властивість Write, що дозволяє клієнту змінювати ці параметри. Водночас, для характеристики статусу було застосовано властивість Notify, що дає змогу клієнту отримувати оновлення про стан з'єднання або внутрішні помилки в режимі реального часу, а Characteristic UUID для рівеня заряду батареї використовує властивість Read. Це,

своєю чергою, оптимізує споживання ресурсів і забезпечує актуальність інформації.

Отже, правильно визначені Properties значно підвищують ефективність та гнучкість взаємодії в BLE-системах, дозволяючи адаптувати обмін даними до специфічних потреб застосунку. Завдяки цим властивостям досягається оптимальний баланс між контролем, прозорістю та енергоспоживанням, що є критично важливим для пристроїв Інтернету речей.

- Descriptors - додаткові метадані, що надають розширену інформацію про характеристику, уточнюючи її призначення або налаштування. Ці елементи дозволяють додавати контекст до характеристик, наприклад, вказувати одиниці вимірювання, діапазон значень або опис для зручності розробника. Таким чином, Descriptors сприяють покращенню читабельності та зрозумілості BLE-сервісів, що, своєю чергою, спрощує інтеграцію та налагодження.

Зокрема, Descriptors можуть використовуватися для конфігурації нотифікацій, як-от BLE2902, що є Client Characteristic Configuration Descriptor. Цей дескриптор дозволяє клієнту вмикати або вимикати сповіщення про зміни значень характеристики, що є важливим для асинхронного обміну даними та реакції системи на події. Його наявність забезпечує можливість динамічного налаштування поведінки характеристики відповідно до потреб клієнтського застосунку.

Таким чином, Descriptors відіграють значну роль у деталізації та конфігурації характеристик BLE, надаючи додатковий рівень інформації, що є корисним для розробників. Вони дають змогу створювати більш інформативні та гнучкі інтерфейси, що, своєю чергою, підвищує зручність використання та потенціал для розширення функціоналу системи. Наявність таких метаданих значно полегшує процес інтеграції та розуміння архітектури BLE-пристрою.

З огляду на зазначене, BLE є оптимальним вибором для створення мобільного клієнтського додатка, що керує побутовим освітленням із мінімальними затратами енергії та без необхідності підтримання довготривалого з’єднання.

1.2. Платформи Arduino з підтримкою BLE

Плати Arduino широко використовуються в сучасних рішеннях в сфері досліджень Інтернету речей (IoT), зокрема в сфері розумного будинку та розумного міста [9, 13, 25].

У межах рішення на базі мікроконтролерів платформи Arduino особливу увагу слід приділити моделям з інтегрованим модулем Bluetooth Low Energy. Поширеними варіантами є:

1. Arduino Nano 33 BLE - плата, яка є компактним рішенням, основу якого становить процесор nRF52840 (ARM Cortex-M4). Зазначений процесор забезпечує апаратну підтримку стандарту BLE 5.0, що, своєю чергою, гарантує високу ефективність передачі даних та низьке енергоспоживання. Крім того, наявність апаратного шифрування підвищує безпеку комунікацій, що є важливим аспектом для пристроїв Інтернету речей.

Плата також оснащена вбудованим датчиком руху, що розширює її функціональні можливості для інтеграції в різноманітні проєкти. Підтримка Arduino IDE значно спрощує процес розробки та налагодження програмного забезпечення, що робить її доступною для широкого кола розробників.

Таким чином, Arduino Nano 33 BLE є оптимальним вибором для застосунків, де пріоритетними є компактність, низьке енергоспоживання та потреба в базових сенсорних можливостях, що, у свою чергу, відповідає сучасним тенденціям у розробці компактних IoT-пристроїв.[16].

2. Arduino Nano 33 BLE Sense - розширена версія Nano 33 BLE та відрізняється наявністю інтегрованого сенсорного модуля. Сенсорний модуль дозволяє відстежувати такі параметри, як температура, вологість, тиск, рівень освітленості та прискорення. Завдяки цим вбудованим сенсорам реалізуються додаткові сценарії керування, зокрема освітленням, на основі даних про навколишнє середовище.

Це, своєю чергою, забезпечує можливість створення адаптивних систем, що реагують на зміни зовнішніх умов, підвищуючи їхню функціональність та

зручність використання. Актуальність такого рішення зумовлена зростанням попиту на інтелектуальні системи "розумного дому", що вимагають комплексного збору даних про оточення.

Отже, Arduino Nano 33 BLE Sense є доцільним рішенням для проєктів, які потребують розширених можливостей сенсорного введення для автоматизації та адаптації до середовища, що дозволяє створювати більш складні та автономні системи керування.[17].

3. ESP32 DevKitC - хоч технічно не є частиною сімейства Arduino, широко застосовується в Arduino-проєктах завдяки підтримці Arduino Core for ESP32. Процесор ESP32 містить два ядра Tensilica LX6, а також модулі BLE 4.2 та Wi-Fi. Це робить його універсальним рішенням для реалізації завдань локальної автоматизації.

В рамках даного дослідження саме ESP32 DevKitC була обрана як оптимальна платформа. Такий вибір зумовлений оптимальним поєднанням обчислювальних ресурсів, високої енергоефективності та доступності бібліотек, необхідних для реалізації Web Bluetooth API на клієнтській стороні.

ESP32 DevKitC є найбільш прийнятним варіантом для проєктів, які вимагають як BLE-комунікації, так і підключення до Wi-Fi, забезпечуючи високу продуктивність та гнучкість для завдань домашньої автоматизації та Інтернету речей. Це, своєю чергою, дозволило ефективно реалізувати поставлені цілі дослідження.[18].

Отже, вибір платформи залежить від вимог до розміру, енергоспоживання, наявності додаткових сенсорів та необхідності підтримки Wi-Fi. У даному дослідженні обрана плата ESP32 DevKitC за рахунок оптимального поєднання обчислювальних ресурсів, енергоефективності та доступності бібліотек для реалізації Web Bluetooth API клієнтом.

1.3. Web Bluetooth API: можливості та обмеження

Web Bluetooth API є сучасним інструментом для організації безпосереднього взаємодії вебзастосунку з пристроями BLE без проміжного серверного шару. API набуває підтримки в браузерах на основі Chromium (Chrome, Edge) у мобільних та десктопних версіях за умови HTTPS-контексту [3, 15, 22].

Основні можливості Web Bluetooth API:

navigator.bluetooth.requestDevice() – метод, який виконує ініціалізацію пошуку та встановлення з'єднання з BLE-пристроєм, використовуючи фільтри за UUID сервісів. Він дозволяє користувачеві обрати конкретний пристрій зі списку доступних, що відповідають заданим критеріям. Таким чином, забезпечується цільове підключення до необхідного апаратного компонента.

Застосування фільтрів за UUID сервісів є важливим для оптимізації процесу виявлення пристроїв, оскільки це дозволяє відсіяти нерелевантні BLE-пристрої та представити користувачеві лише ті, що сумісні з функціоналом вебдодатка. Це, своєю чергою, значно спрощує користувацький досвід та підвищує ефективність пошуку.

Отже, navigator.bluetooth.requestDevice() є першим і фундаментальним кроком у встановленні комунікації між вебдодатком та BLE-пристроєм. Він забезпечує контрольований та безпечний механізм для початкового підключення, що є основою для подальшої взаємодії.

device.gatt.connect() – метод, який відповідає за встановлення GATT-з'єднання для подальшої роботи з характеристиками пристрою. Після успішного вибору BLE-пристрою за допомогою requestDevice(), необхідно активувати GATT-сервер для доступу до його сервісів та характеристик. Це, своєю чергою, дозволяє розпочати обмін даними.

Встановлення GATT-з'єднання є критично важливим етапом, оскільки саме через нього відбувається структурована взаємодія з пристроєм відповідно до

профілю Generic Attribute Profile (GATT). Без цього з'єднання доступ до даних характеристик, які визначають функціональність пристрою, був би неможливим.

Таким чином, device.gatt.connect() забезпечує логічний міст між ідентифікованим BLE-пристроєм та вебдодатком, відкриваючи шлях для читання, запису та підписки на значення характеристик. Це є необхідною умовою для повноцінного функціонування розробленого комунікаційного інтерфейсу.

getPrimaryService(), getCharacteristic() – методи, які надають доступ до конкретних сервісів та характеристик BLE-пристрою. Після встановлення GATT-з'єднання вони дозволяють вебдодатку ідентифікувати та отримати посилання на необхідні сервіси за їх UUID, а потім — на конкретні характеристики всередині цих сервісів.

Ці методи є основними інструментами для навігації по GATT-архітектурі пристрою. Завдяки їх використанню розробник може програмно звертатися до певних функціональних блоків (сервісів) та їхніх атрибутів (характеристик), що є необхідним для реалізації логіки керування та отримання даних.

Отже, getPrimaryService() та getCharacteristic() забезпечують структурований та адресний доступ до функціональних елементів BLE-пристрою. Це, своєю чергою, дозволяє вебдодатку ефективно взаємодіяти з різними компонентами системи, реалізовуючи складні сценарії керування.

characteristic.readValue(), characteristic.writeValue() - призначені для читання та запису значень характеристик. Завдяки readValue() вебдодаток може отримувати поточний стан певного параметра з BLE-пристрою, наприклад, рівень заряду батареї. writeValue(), своєю чергою, дозволяє відправляти команди або нові значення на пристрій, такі як зміна яскравості або температури освітлення.

Ці операції є базовими для реалізації двостороннього обміну даними між клієнтом та BLE-сервером. Вони забезпечують безпосередній контроль над функціями пристрою та дозволяють вебдодатку як отримувати зворотний зв'язок, так і надсилати керуючі сигнали.

Таким чином, characteristic.readValue() та characteristic.writeValue() є ключовими для реалізації функціональності керування та моніторингу в системі, що, у свою чергу, забезпечує інтерактивність та ефективність взаємодії.

characteristic.startNotifications() - дозволяє підписатися на оновлення значень характеристики з подальшою обробкою в колбеках. Ця функціональність є особливо важливою для відстеження змін стану пристрою в режимі реального часу без необхідності постійного опитування. Завдяки цьому забезпечується асинхронна передача даних.

При активації нотифікацій BLE-пристрій автоматично надсилає сповіщення вебдодатку щоразу, коли значення певної характеристики змінюється. Це дозволяє оперативно реагувати на події, такі як втрата з'єднання або зміна внутрішніх помилок, без зайвого навантаження на систему.

Отже, characteristic.startNotifications() значно підвищує реактивність та енергоефективність взаємодії, оскільки вебдодаток отримує інформацію лише тоді, коли вона стає актуальною. Це, своєю чергою, є важливим для оптимізації роботи як клієнтської, так і серверної частини системи. Незважаючи на значні переваги, API має обмеження:

- підтримка лише у браузерах Chromium в умовах HTTPS: Підтримка лише у браузерах Chromium в умовах HTTPS: Підтримка Web Bluetooth API наразі обмежена браузерами на основі Chromium, до яких належать Chrome та Edge. Це означає, що вебдодатки, які використовують цей API, не будуть коректно функціонувати в інших популярних браузерах, таких як Firefox або Safari (за винятком iOS), що, своєю чергою, обмежує кросбраузерну сумісність. Більше того, для забезпечення функціонування вимагається обов'язкове використання HTTPS-контексту, що є вимогою безпеки для доступу до чутливих апаратних ресурсів пристрою. Це, своєю чергою, вимагає від розробників забезпечення захищеного з'єднання, що може ускладнити процес розгортання та тестування на початкових етапах.

Таким чином, обмежена підтримка браузерами та вимога HTTPS є суттєвими факторами, що впливають на доступність та розгортання рішень на базі Web Bluetooth API.

- потребує активного стану вкладки: функціональність у фоні обмежена політикою браузера: функціональність Web Bluetooth API у фоновому режимі є обмеженою через політику браузера. Це, по суті, означає, що для безперервної взаємодії з BLE-пристроєм необхідний активний стан вкладки. Таке обмеження може стати проблемою для застосунків, які потребують постійного моніторингу або керування у фоновому режимі, таких як системи домашньої автоматизації або медичні пристрої, де безперебійний зв'язок є критично важливим.

Відтак, для застосунків, що вимагають тривалого або постійного фонового зв'язку, необхідно передбачати альтернативні механізми або інформувати користувача про необхідність підтримки активної вкладки.

Це обмеження зумовлює необхідність ретельного планування архітектури вебдодатків, які покладаються на Web Bluetooth API для безперервної фонової взаємодії.

- відсутність прямої підтримки Web Bluetooth у iOS (Safari): Зокрема, у iOS відсутня пряма підтримка Web Bluetooth API у браузері Safari. Це, своєю чергою, унеможливлює використання розробленого рішення на пристроях Apple без додаткових обхідних шляхів або нативних застосунків. Це є значним обмеженням для розробників, оскільки обмежує потенційну аудиторію користувачів.

Відтак, для забезпечення кросплатформної сумісності та охоплення користувачів iOS, необхідно розглядати розробку окремих нативних мобільних застосунків або використовувати проміжне серверне рішення.

Таким чином, відсутність нативної підтримки на iOS є критичним фактором, що вимагає додаткових зусиль для розробки універсальних рішень.

- обмеження на кількість одночасних підключень до пристроїв: Існують обмеження на кількість одночасних підключень до пристроїв. Це, своєю чергою, може впливати на масштабованість системи, якщо передбачається

керування великою кількістю BLE-пристроїв одночасно, наприклад, у системах "розумного будинку" з численними світловими приладами.

Ці обмеження можуть бути пов'язані як з можливостями самого браузера, так і з апаратними особливостями операційної системи, що, у свою чергу, потребує оптимізації архітектури системи для багатопристроєвої взаємодії.

Отже, планування системи, що передбачає одночасне керування кількома BLE-пристроями, має враховувати ці ліміти для забезпечення стабільної роботи.

- затримки при першому скануванні та при встановленні зв’язку: Слід зважати на можливі затримки при першому скануванні пристроїв та під час встановлення зв'язку, які можуть становити до 2–3 секунд. Це, своєю чергою, може негативно впливати на користувацький досвід, особливо в застосунках, де вимагається миттєва реакція на дії користувача.

Такі затримки можуть бути спричинені процесами виявлення пристроїв, встановлення GATT-з'єднання та обміну початковими даними, що, у свою чергу, вимагає оптимізації алгоритмів підключення.

Таким чином, для мінімізації негативного впливу на користувацький досвід, необхідно передбачати механізми візуального зворотного зв'язку або індикації стану підключення під час цих затримок.

Отже, Web Bluetooth API забезпечує достатній набір методів для реалізації CRUD-операцій над GATT-сервісами, але потребує врахування платформи та умов виконання вебдодатка.

1.4. Огляд існуючих комунікаційних інтерфейсів

Існує кілька комунікаційних інтерфейсів для керування BLE-пристроями за допомогою вебінтерфейсів та мобільних додатків [2, 7]. Зокрема, популярними рішеннями є:

– Bluestack Connect - комерційна платформа для управління BLE-пристроями через веб та мобільні додатки з підтримкою кастомізованих GATT-сервісів [20].

- WebBLE.js - open-source бібліотека, що спрощує роботу з Web Bluetooth API, надаючи обгортки для пошуку пристроїв та обміну даними.
- nRF Connect for Mobile - мобільний застосунок від Nordic Semiconductor для відлагодження та взаємодії з BLE-пристроями, який може експортувати налаштування та скрипти [14].
- Bluepy & Flask - Python-основа серверного додатка з Bluepy для роботи з BLE та Flask як вебінтерфейс, що забезпечує проміжне програмне рішення [5, 10, 11, 19].

Таким чином, більшість існуючих інструментів або вимагають використання власного серверного компонента, або обмежені лише низькорівневими API без зручного вебінтерфейсу. У порівнянні із запропонованим клієнтським рішенням, що працює виключно в браузері без сторонніх серверів, ці системи мають меншу автономність та підвищені вимоги до розгортання.

1.5. Висновки за розділом

Таким чином, проведений аналіз предметної області дозволив сформулювати ключові висновки, що стали підґрунтям для подальшої розробки системи.

По-перше, встановлено, що Bluetooth Low Energy (BLE) є оптимальним протоколом для енергоефективного керування побутовими пристроями, особливо в умовах передачі невеликих обсягів даних. Це зумовлено його низьким енергоспоживанням, що є критично важливим для автономних пристроїв, а також широким поширенням у пристроях Інтернету речей (IoT) та системах домашньої автоматизації. Отже, вибір BLE як основи комунікаційного інтерфейсу є повністю обґрунтованим з погляду ефективності та сучасності.

По-друге, серед різноманіття платформ з підтримкою BLE, плата ESP32 DevKitC виділяється оптимальним поєднанням обчислювальної потужності, універсальності інтерфейсів та низького енергоспоживання. По суті, ESP32 DevKitC, незважаючи на те, що технічно не є частиною сімейства Arduino, активно застосовується в Arduino-проєктах завдяки підтримці Arduino Core for ESP32. Ця платформа включає два ядра Tensilica LX6, BLE 4.2 та Wi-Fi, що робить її багатофункціональним рішенням для завдань локальної автоматизації. Це, своєю чергою, дозволило забезпечити надійну апаратну базу для реалізації проєкту.

По-третє, Web Bluetooth API забезпечує всі необхідні методи для встановлення GATT-з'єднання та виконання CRUD-операцій (створення, читання, оновлення, видалення) над характеристиками. Однак, слід мати на увазі, що для його функціонування потрібен HTTPS-контекст та підтримка з боку браузера, що обмежує його застосування переважно браузерами на основі Chromium (Chrome, Edge) у мобільних та десктопних версіях. Таким чином, попри певні обмеження, Web Bluetooth API надає достатній інструментарій для розробки клієнтського рішення.

Нарешті, аналіз наявних комерційних та open-source рішень для керування BLE-пристроями показав, що вони здебільшого залежать від серверних компонентів або характеризуються відсутністю зручного клієнтського інтерфейсу. Це, своєю чергою, створює додаткові точки відмови та збільшує затримки при обміні даними, що є недоліком у порівнянні з прямим зв’язком. Отже, відсутність повноцінного клієнтського рішення, яке інтегрує Web Bluetooth API з автоматизацією за сонячним циклом без проміжних серверів, обґрунтовує нагальну необхідність розробки власної системи керування освітленням у рамках цього дослідження.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Постановка технічного завдання

Технічне завдання (ТЗ) визначає ключові функціональні та експлуатаційні вимоги до системи керування LED-освітленням через веббраузер з використанням BLE без проміжного серверного компонента.

Функціональні вимоги:

1. Керування яскравістю (0–100%) та колірною температурою (від теплого до холодного білого) двоканальної LED-стрічки;

2. Підтримка режимів освітлення: постійний, стробоскоп, імітація вогню, імітація блискавки;

3. Вмикання та вимикання освітлення за допомогою кнопки на вебінтерфейсі;

4. Автоматичне планування увімкнення/вимкнення на основі часу сходу і заходу сонця із врахуванням геолокації користувача [8, 12];

5. Відображення стану з’єднання та поточних параметрів у вебдодатку;

6. Підтримка повторного автоматичного перепідключення у разі втрати BLE-з’єднання.

Нефункціональні вимоги:

— Сумісність браузерів на базі Chromium (Chrome, Edge) у версіях, що підтримують Web Bluetooth API;

— Робота в умовах HTTPS-контексту;

— Мінімальне затримення керувальних команд (не більше 100 мс) при активному з’єднанні;

— Відмовостійкість: можливість відновлення з’єднання без перезавантаження сторінки;

— Простота масштабування: можливість підключення до декількох пристроїв одночасно (до 3);

— Зручність користувацького інтерфейсу: адаптивний дизайн для мобільних пристроїв.

Таким чином, сформульовані вимоги визначають межі проєктування та слугують орієнтиром для подальших етапів розробки системи.

2.2. Функціональні та нефункціональні вимоги

Функціональні вимоги визначають поведінку системи з точки зору користувача, а нефункціональні - характеристики якості та експлуатаційні обмеження.

– Функціональні вимоги:
– Ініціалізація BLE-з’єднання з ESP32 через Web Bluetooth API;
– Регулювання яскравості за допомогою повзунка (0–255);
– Зміна колірної температури через управління двома PWM-каналами;
– Вибір режиму освітлення (normal, strobo, fire, lightning) через селектор;
– Кнопка вмикання/вимикання освітлення з індикацією стану;
– Отримання координат користувача через Geolocation API;
– Виклик зовнішнього API для розрахунку часу сходу/заходу сонця;
– Автоматичне встановлення таймерів на вмикнення/вимкнення за обчисленими часами;
– Відображення актуального статусу автоматизації та параметрів освітлення.

Нефункціональні вимоги:

– Сумісність з мобільними браузерами на базі Chromium у мобільних та десктопних версіях;
– Робота лише в умовах HTTPS через необхідність доступу до Geolocation та Web Bluetooth;
– Час реагування інтерфейсу на події не більше 100 мс при активному підключенні;

- Можливість одночасної роботи з кількома пристроями (до 3) без значного зниження продуктивності;
- Легка масштабованість UI шляхом використання компонентного підходу в JavaScript;
- Забезпечення відмовостійкості шляхом обробки подій gattserverdisconnected та автоматичного перепідключення;
- Адаптивний дизайн інтерфейсу для різних розмірів екранів із використанням CSS media queries.

Таким чином, чітко визначені функціональні та нефункціональні вимоги закладають фундамент для розробки надійної, ефективної та користувацько-зручної системи керування LED-освітленням, яка відповідатиме сучасним стандартам веброзробки та бездротових комунікацій.

2.3. Архітектура апаратного рівня

Апаратна архітектура системи базується на мікроконтролері ESP32 DevKitC, що забезпечує підтримку Bluetooth Low Energy та достатню обчислювальну потужність для керування двоканальною LED-стрічкою.

Основні компоненти апаратного рівня:

- ESP32 DevKitC - центральний контролер системи. Ця плата оснащена двоядерним процесором Tensilica LX6, який забезпечує достатню обчислювальну потужність для керування двоканальною LED-стрічкою. Важливою особливістю ESP32 є інтегрована підтримка BLE 4.2 та Wi-Fi, що робить її універсальним рішенням для завдань локальної автоматизації.

  Застосування ESP32 DevKitC в середовищі Arduino можливе завдяки підтримці Arduino Core for ESP32. Це дозволяє використовувати звичні інструменти розробки, спрощуючи процес програмування та налагодження. З огляду на зазначене, дана платформа є оптимальним

поєднанням обчислювальних ресурсів, енергоефективності та доступності бібліотек для реалізації Web Bluetooth API клієнтом.

Таким чином, ESP32 DevKitC є надійною основою для апаратної частини системи, що забезпечує необхідну функціональність та гнучкість для інтеграції з вебдодатком.

– MOSFET-драйвери (IRLZ44N) - для керування LED-стрічкою використовуються MOSFET-драйвери моделі IRLZ44N. Кожен канал LED-стрічки (теплий та холодний) підключається через окремий MOSFET-драйвер. Ці драйвери є критично важливими компонентами, що дозволяють регулювати PWM-сигнали (широтно-імпульсна модуляція) з ESP32.

  Регулювання PWM-сигналів за допомогою MOSFET-драйверів дає змогу точно змінювати яскравість та спектр світла LED-стрічки. Це забезпечує плавне керування інтенсивністю освітлення та колірною температурою. Завдяки цьому досягається висока точність налаштування параметрів світла, що, у свою чергу, є однією з ключових функціональних вимог системи.

  Отже, MOSFET-драйвери є необхідними для забезпечення ефективного та точного керування живленням LED-стрічки, забезпечуючи перетворення низьковольтних керуючих сигналів ESP32 на відповідні рівні потужності для освітлювальних елементів.

– LED-стрічка 12 В (двоканальна) - система передбачає використання двоканальної LED-стрічки, яка живиться від джерела напруги 12 В. Ця стрічка складається з діодів, що забезпечують можливість зміни балансу між теплим та холодним світінням. Така функціональність дозволяє користувачеві налаштовувати колірну температуру освітлення відповідно до власних потреб.

  Двоканальна структура LED-стрічки є основою для реалізації регулювання колірної температури, що, у свою чергу, відрізняє дану систему від простіших рішень з одноканальним освітленням. Завдяки

цьому досягається підвищена гнучкість у налаштуванні освітлення, забезпечуючи комфортне візуальне середовище.

Таким чином, двоканальна LED-стрічка є основним виконавчим елементом системи, що безпосередньо забезпечує світловий потік з можливістю тонкого налаштування його характеристик.

– Живлення 12 В / 5 А - для стабільної роботи LED-стрічки та мікроконтролера ESP32 передбачено використання джерела живлення 12 В / 5 А. Джерело постійного струму 12 В забезпечує безпосереднє живлення LED-стрічки. Це є важливим для підтримки необхідної яскравості та стабільності роботи світлодіодів.

  Крім того, це ж джерело живлення через інтегрований стабілізатор на платі (5 В Reg) подає напругу 5 В на VIN ESP32. Це забезпечує стабільну роботу самого контролера, що є критично важливим для безперебійного функціонування програмного забезпечення та BLE-комунікації.

  Отже, правильно підібране та реалізоване живлення є фундаментальним аспектом, що гарантує надійність та довготривалість роботи всієї апаратної системи.

– Схема з’єднань - в апаратній архітектурі системи передбачено спільну шину GND (земля) для всіх компонентів. Це включає ESP32 DevKitC, LED-стрічку та джерело живлення. Об'єднання всіх компонентів на спільній шині GND є важливим для уніфікації посилення сигналів та мінімізації електричних перешкод, що, своєю чергою, підвищує стабільність роботи системи.

  GPIO15 та GPIO2 мікроконтролера ESP32 підключені до затворів MOSFET-драйверів для каналів warm і cool відповідно. Це дозволяє ESP32 генерувати PWM-сигнали, які, проходячи через MOSFET-драйвери, регулюють струм, що надходить до відповідних каналів LED-стрічки.

Таким чином, така схема з'єднань забезпечує простоту монтажу, надійність взаємодії між компонентами та можливість швидкого обслуговування системи.

Пропонуємо наступну електричну схему підключення на рис. 2.1. В схемі показано розташування контактів ESP32, MOSFET і монтажні з’єднання. Така архітектура забезпечує простоту монтажу, надійність з’єднань та можливість швидкого обслуговування.

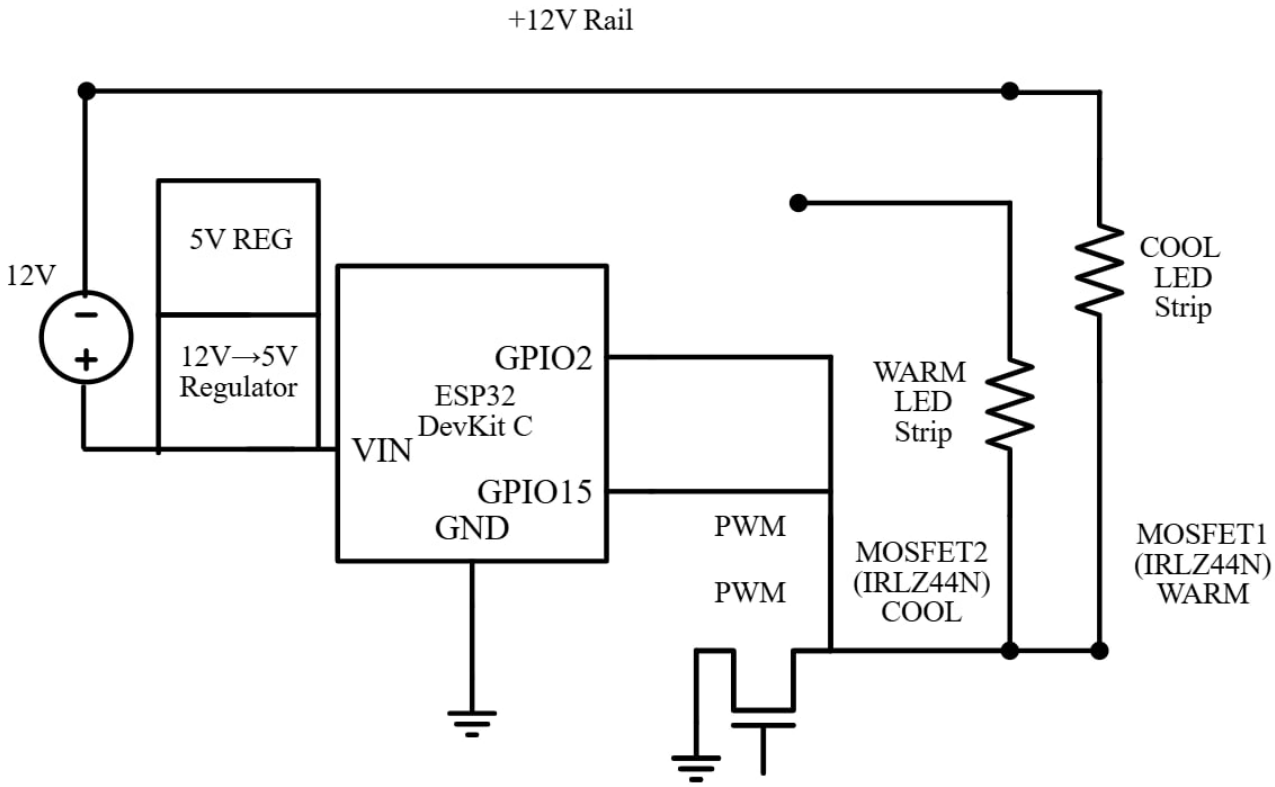


Рисунок 2.1 – Електрична схема підключення

Представлена на рисунку 2.1. електрична схема є результатом поєднання вимог щодо простоти монтажу, енергоефективності та надійності роботи системи керування двоканальною LED-стрічкою на базі мікроконтролера ESP32.

Основні принципи побудови схеми:

1. Живлення: джерело постійного струму 12 В забезпечує безпосереднє живлення LED-стрічки. Крім того, через інтегрований стабілізатор на платі (5 В Reg) напруга 5 В подається на VIN ESP32, що є критично

важливим для стабільної роботи контролера. Такий підхід забезпечує надійне живлення як освітлювальних елементів, так і керуючого мікроконтролера.

2. Розподіл каналів освітлення: Два канали LED-стрічки — теплий та холодний білий — підключені через окремі N-канальні MOSFET (IRLZ44N). Це дозволяє за допомогою PWM-сигналів (широтно-імпульсної модуляції) з GPIO15 та GPIO2 регулювати як загальну яскравість, так і баланс між цими каналами. Таким чином, досягається гнучке налаштування колірної температури освітлення.
3. Логіка керування: GPIO15 мікроконтролера призначено для теплого каналу (WARM), а GPIO2 — для холодного (COOL). PWM-сигнали, що генеруються ESP32, проходять через затвори MOSFET, формуючи керуючі сигнали. Ці сигнали подаються на мінусові виводи LED-стрічок, тоді як позитивний полюс під'єднаний до +12 V Rail. Отже, реалізується точне та незалежне керування кожним каналом.
4. З'єднання «землі»: усі компоненти системи, включаючи GND ESP32, витратну шину LED-стрічок та джерело живлення, мають спільну шину GND. Це є важливим для уніфікованого посилення сигналів, що, своєю чергою, мінімізує вплив електричних перешкод. Таким чином, забезпечується стабільність та надійність електричних з'єднань.
5. Захисні елементи: у місцях переходу з PWM-сигналу на потужний струм передбачено встановлення конденсатора на GND після MOSFET. Цей конденсатор призначений для згладжування перехідних процесів та зменшення електромагнітних перешкод, що, своєю чергою, підвищує стабільність роботи системи. Зазначені захисні елементи забезпечують довговічність та безпеку функціонування апаратної частини.

Таким чином, спроектована схема реалізує вимоги до безпечного та ефективного керування двоканальною LED-стрічкою, забезпечує ізоляцію високої та низької напруги та підтримує точне налаштування яскравості та кольорової температури.

2.4. Проєктування BLE-GATT-сервісів і характеристик

Архітектура програмного рівня на стороні ESP32 базується на моделі BLE GATT-сервера, який надає клієнту набір характеристик для керування освітленням [6].

Основні сервіси та характеристики:

1. Сервіс керування освітленням (UUID: 12345678-1234-1234-1234-1234567890ab)

Цей сервіс є центральним компонентом системи, призначеним для контролю основних параметрів LED-освітлення. Він включає чотири основні характеристики, кожна з яких відповідає за певну функцію керування. Використання унікального UUID для сервісу забезпечує його однозначну ідентифікацію клієнтським пристроєм, що, своєю чергою, дозволяє ефективно встановлювати зв'язок та обмінюватися даними.

– CHAR_BRIGHTNESS (UUID: abcd0001-1234-5678-1234-56789abcdef0, Write) – характеристика, що призначена для регулювання загальної яскравості освітлення в діапазоні від 0 до 255. Вона має властивість "Write", що дозволяє клієнтському пристрою записувати нове значення яскравості на ESP32. Це, своєю чергою, забезпечує можливість плавного налаштування інтенсивності світла відповідно до потреб користувача. Метод writeValue() використовується для передачі однобайтового значення (0–255), яке зберігається у змінній brightness на мікроконтролері. Після оновлення значення викликається функція applyPWM(), яка розподіляє яскравість між каналами. Таким чином, реалізується динамічне керування інтенсивністю світіння.

Отже, CHAR_BRIGHTNESS є ключовим елементом для контролю світлового потоку, що, у свою чергу, підвищує функціональність системи.

– CHAR_TEMPERATURE (UUID: abcd0002-1234-5678-1234-56789abcdef0, Write) – характеристика, яка відповідає за встановлення балансу між каналами теплого та холодного білого світла. Вона також має властивість "Write", що дозволяє вебдодатку змінювати колірну температуру LED-стрічки.

Це, своєю чергою, забезпечує гнучке налаштування відтінку освітлення, імітуючи різні джерела світла.

Значення, отримане цією характеристикою (байт для балансування каналів), призначається змінній temperature на ESP32. Функція applyPWM() потім використовує це значення для розрахунку відповідних значень для теплого та холодного каналів, контролюючи їх за допомогою PWM. Це, по суті, забезпечує тонке керування спектром світла.

Таким чином, CHAR_TEMPERATURE є необхідним для реалізації динамічної зміни колірної температури, що, у свою чергу, підвищує комфорт та адаптивність системи освітлення.

– CHAR_MODE (UUID: abcd0003-1234-5678-1234-56789abcdef0, Write) - використовується для вибору режиму освітлення, який може бути "normal", "strobo", "fire" або "lightning". Завдяки властивості "Write" клієнтський додаток може надсилати відповідну команду для активації бажаного світлового ефекту. Це, своєю чергою, дозволяє системі переходити між статичним освітленням та динамічними ефектами.

Прошивка ESP32 обробляє рядкову команду, отриману через цю характеристику, та зберігає її у змінну mode. У головному циклі loop() мікроконтролер викликає відповідну функцію ефекту (наприклад, effectStrobo(), effectFire(), effectLightning()) залежно від значення цієї змінної.

Отже, CHAR_MODE забезпечує гнучке керування світловими ефектами, що, у свою чергу, розширює функціональні можливості системи.

– CHAR_POWER (UUID: abcd0004-1234-5678-1234-56789abcdef0, Write) - призначена для керування вмиканням та вимиканням освітлення. Вона також має властивість "Write", дозволяючи вебдодатку перемикати стан живлення LED-стрічки. Це є основним методом для ввімкнення/вимкнення всієї системи освітлення.

Ця характеристика приймає бінарне значення (0 або 1), яке керує змінною powerOn на мікроконтролері. Залежно від стану powerOn, функція applyPWM()

повністю вмикає або вимикає освітлення, встановлюючи нульові значення для PWM-каналів при powerOn=false.

Таким чином, CHAR_POWER є фундаментальною характеристикою для базового керування живленням освітлення, що, у свою чергу, забезпечує простий та ефективний спосіб ввімкнення/вимкнення системи.

2. Сервіс діагностики (UUID: 87654321-4321-4321-4321-ba0987654321) Сервіс діагностики призначений для інформування клієнтського пристрою про внутрішній стан системи та можливі помилки. Він забезпечує прозорість роботи мікроконтролера ESP32, що є важливим для своєчасного виявлення та усунення несправностей. Використання окремого UUID для цього сервісу дозволяє чітко відокремити діагностичні дані від керуючих, що, своєю чергою, спрощує архітектуру та підвищує модульність системи.

– CHAR_STATUS (UUID: dcba0001-4321-8765-4321-0fedcba98765, Notify) - інформує клієнта про поточний стан з’єднання та внутрішні помилки. Вона має властивість "Notify", що дозволяє вебдодатку отримувати асинхронні сповіщення про зміни стану без необхідності постійного опитування. Це, своєю чергою, оптимізує використання ресурсів та забезпечує оперативне реагування на події.

Для отримання оновлень статусу клієнт здійснює підписку на CHAR_STATUS через метод startNotifications(). Змінені значення, передані мікроконтролером, обробляються у відповідних колбеках, що дозволяє вебдодатку динамічно відображати актуальну інформацію про з'єднання та помилки. Це, по суті, забезпечує постійний моніторинг функціональності системи.

Отже, CHAR_STATUS є важливим інструментом для забезпечення надійності та прозорості роботи системи, дозволяючи користувачеві бути поінформованим про будь-які зміни в її стані.

– CHAR_BATTERY (UUID: dcba0002-4321-8765-4321-0fedcba98765, Read) - повертає рівень заряду від джерела живлення в діапазоні від 0 до 100%. Вона має властивість "Read", що дозволяє клієнтському пристрою запитувати поточне значення заряду за потреби. Це, своєю чергою, є важливим для

моніторингу автономності системи, особливо якщо вона живиться від акумулятора.

Читання рівня заряду батареї виконується за запитом через метод readValue(). Отримане значення може бути відображено у користувацькому інтерфейсі, що надає користувачеві актуальну інформацію про енергетичний стан пристрою. Таким чином, забезпечується можливість планування використання системи та її своєчасного обслуговування.

Таким чином, CHAR_BATTERY є корисною характеристикою для інформування користувача про стан живлення пристрою, що підвищує його експлуатаційну зручність.

Процедури взаємодії:

Взаємодія між вебклієнтом та BLE GATT-сервером на ESP32 відбувається за чітко визначеними процедурами, що забезпечують ефективний обмін даними та керування системою освітлення.

– При підключенні клієнт виконує пошук сервісу та характеристик через getPrimaryService() та getCharacteristic(): При встановленні з'єднання, після успішного виявлення пристрою, клієнтський вебдодаток ініціює пошук основного сервісу керування освітленням, використовуючи його Service UUID. Після цього, за допомогою getCharacteristic(), відбувається отримання доступу до всіх необхідних характеристик (таких як яскравість, температура, режим, живлення, статус та батарея) за їхніми унікальними UUID. Це, своєю чергою, забезпечує структурований доступ до функціональних елементів пристрою.

– Для регулювання яскравості та температури використовується метод writeValue() з відповідними Uint8Array: Для керування яскравістю та колірною температурою LED-стрічки клієнтський вебдодаток використовує метод writeValue(). Значення параметрів передаються у вигляді об'єктів Uint8Array, що дозволяє мікроконтролеру ESP32 безпосередньо інтерпретувати отримані дані та застосовувати їх для керування PWM-сигналами. Це, своєю чергою, забезпечує точне та оперативне налаштування освітлення.

– Для отримання оновлень статусу підписка на CHAR_STATUS здійснюється через startNotifications(), змінені значення обробляються у колбеках: З метою моніторингу поточного стану системи та внутрішніх помилок, вебдодаток підписується на характеристику CHAR_STATUS за допомогою методу startNotifications(). При зміні значення цієї характеристики, BLE-сервер автоматично надсилає сповіщення, які обробляються у відповідних JavaScript-колбеках на стороні клієнта. Це, своєю чергою, дозволяє забезпечити зворотний зв'язок у реальному часі та підвищити реактивність інтерфейсу.

– Читання рівня заряду батареї виконується за запитом через readValue(): Отримання інформації про рівень заряду батареї, якщо така характеристика підтримується пристроєм, виконується за запитом через метод readValue(). Ця операція є одноразовим читанням, що дозволяє отримати актуальне значення параметра на момент запиту.

Таким чином, структуроване проєктування GATT-сервісів із чітким розмежуванням функцій керування та діагностики забезпечує гнучкість розширення системи новими характеристиками та високий рівень надійності обміну даними.

2.5. Проєктування користувацького інтерфейсу вебдодатка та алгоритму користування вебсистемою

Інтерфейс вебдодатка розроблено з урахуванням принципів адаптивного дизайну та інфостилю для забезпечення зручності керування на мобільних пристроях.

Компоненти інтерфейсу:

6. Кнопка "Підключити" - ініціює виклик `navigator.bluetooth.requestDevice()` для пошуку ESP32 за UUID сервісу;
7. Індикатор стану з’єднання - відображається кольоровою крапочкою та текстом (не підключено, підключення, підключено);

8. Повзунок яскравості - HTML `<input type="range">` зі значенням 0–255 та градієнтом, що ілюструє зміну інтенсивності;
9. Повзунок температури - `<input type="range">` зі шкалою від теплого до холодного білого, супроводжується підписами «Теплий» та «Холодний»;
10. Селектор режимів - `<select>` з опціями normal, strobo, fire, lightning;
11. Кнопка вмикання/вимикання - `<button>` із змінюваним значком лампочки та станом aria-pressed;
12. Чекбокс автоматизації - checkbox для активації планування за сонячним циклом;
13. Кнопка поновлення геолокації - іконка глобуса, що викликає Geolocation API;
14. Віджет повідомлень - блок для відображення повідомлень про помилки та статуси (aria-live="polite").

Особливості реалізації:

– Використання CSS-перемикачів та кастомних стилів для забезпечення однорідного вигляду в темній та світлій темах;
– Гнучка верстка з Flexbox та медіа-запитами для адаптації під різні розміри екранів;
– Забезпечення доступності (ARIA-атрибути, фокусні стані елементів);
– Розділення логіки UI та бізнес-логіки в JavaScript шляхом використання модульного підходу.

Загальний вигляд вебдодатку з усіма вище перерахованими елементами схематично зображено на рис. 2.2.

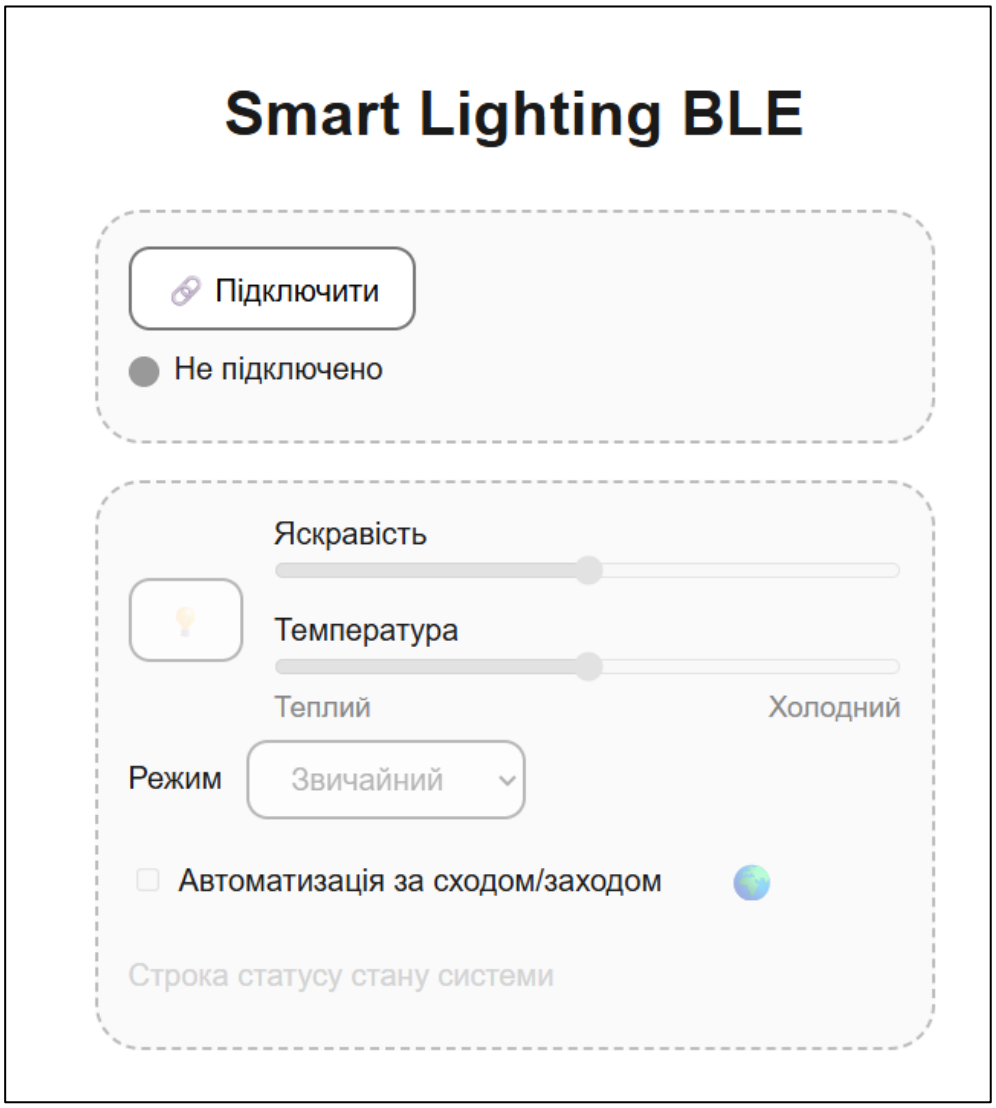


Рисунок 2.2 – Загальний вигляд користувацького інтерфейсу

Пропонується використовувати наступний алгоритм користувацького досвіду для вебсистеми:

1. Початок. На початку роботи користувач відкриває веб-сторінку системи керування освітленням. Виконується перевірка підтримки Bluetooth Low Energy (BLE) у браузері та ініціалізація необхідних UI-компонентів для обміну з мікроконтролером ESP32. У разі відсутності підтримки BLE генерується повідомлення про неможливість подальшої взаємодії.
2. З’єднання. Користувач натискає кнопку «Підключити», що ініціює процедуру сканування BLE-пристроїв. Після вибору ESP32 запускається встановлення захищеного з’єднання. У разі успіху

інтерфейс переходить у режим керування, а в разі відмови або помилки зв’язку залишається в очікуванні з можливістю повторної спроби.

3. Керування освітленням. Після встановлення з’єднання надається доступ до регулювання параметрів освітлення: інтенсивність світла задається слайдером від мінімуму до максимуму, колірна температура - слайдером від теплого до холодного відтінку, а селектор режимів дозволяє обирати між безперервним світлом, стробоскопом, імітацією полум’я та ефектом блискавки. Окремою кнопкою реалізується одночасне вмикання чи вимикання всіх каналів.
4. Автоматизація. При активації чекбокса «Автоматизація за сходом/заходом» система запитує дозвіл на визначення геолокації, потім через зовнішній API отримує точні часи сходу та заходу сонця. На підставі цих даних автоматично формуються таймери для вмикання й вимикання освітлення відповідно до природного циклу.
5. Обробка подій. Усі налаштування, кориговані користувачем, миттєво передаються по BLE до ESP32. Мікроконтролер інтерпретує команди й реалізує зміни: регулює яскравість, колірну температуру або запускає спеціальні світлові ефекти, при цьому забезпечуючи підтвердження успішного виконання.
6. Кінець. Для завершення сеансу користувач натискає «Відключити» або закриває сторінку. Збережені локально в браузері параметри автоматизації застосовуватимуться під час наступних сеансів без необхідності повторного налаштування. Блок-схема алгоритму зображена на рисунку 2.3.

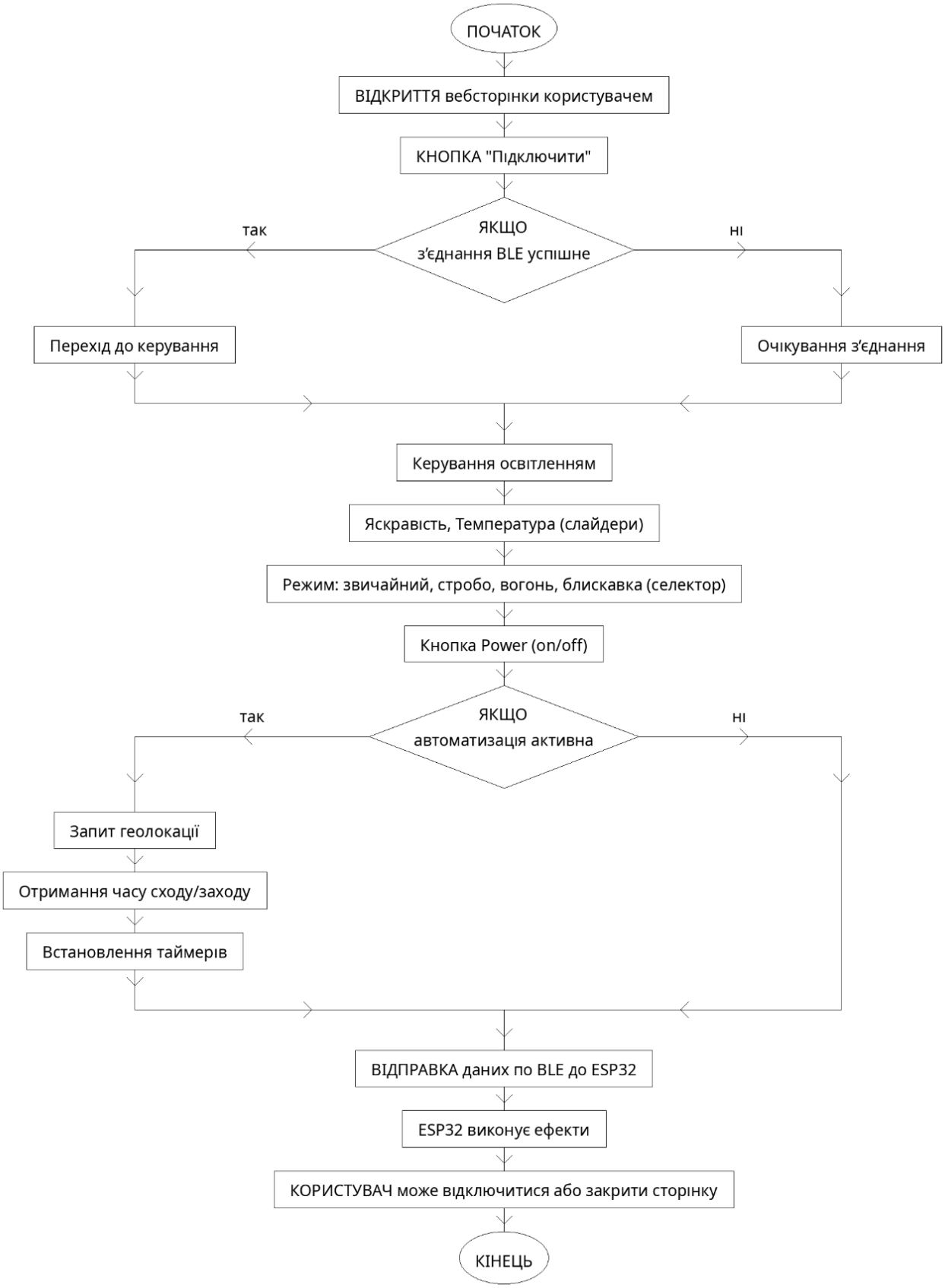


Рисунок 2.3 – Блок-схема алгоритму користувацького досвіду

Таким чином, спроєктований користувацький інтерфейс відповідає вимогам роботи на мобільних пристроях і забезпечує інтуїтивне керування всіма функціями системи.

2.6. Висновки за розділом

Таким чином, внаслідок проведеного проєктування системи було досягнуто узгодження вимог із реальними можливостями платформи ESP32 DevKitC та Web Bluetooth API.

Визначено ключові функціональні та нефункціональні вимоги, які формують основу для реалізації енергоефективного та відмовостійкого рішення керування LED-освітленням.

Розроблена апаратна архітектура забезпечує надійну інтеграцію компонентів із чітким визначенням каналів керування PWM.

Спроєктовані BLE-GATT-сервіси та характеристики гарантують гнучке розширення функціоналу та зручну взаємодію вебклієнта з пристроєм.

Спроєктований користувацький інтерфейс відповідає принципам адаптивності та доступності, що сприятиме позитивному користувацькому досвіду.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ

3.1. Розробка прошивки Arduino

Прошивка ESP32 реалізована мовою C++ із використанням Arduino Core for ESP32 та бібліотек BLEDevice, BLEServer, BLEUtils і BLE2902. У файлі main.ino організовано чітку структуру коду, що складається з трьох основних блоків: ініціалізація, обробка запису характеристик та виконання ефектів у циклі.

У блоці ініціалізації (setup()):

конфігуруються два канали PWM для виводів GPIO15 (теплий) та GPIO2 (холодний) з частотою 5000 Гц і розрядністю 8 біт;

створюється BLE-сервер із сервісом UUID="12345678-1234-1234-1234-1234567890ab" і чотирма характеристиками: `brightness`, `temperature`, `mode` та `power`;

для кожної характеристики встановлюються колбеки на подію запису (`onWrite`), що реалізовано в окремому класі CharacteristicCallback, де при надходженні нових даних оновлюються відповідні змінні.

У блоці обробки запису характеристик:

`CHAR_BRIGHTNESS` приймає однобайтове значення 0–255 і зберігає його в змінну brightness;

`CHAR_TEMPERATURE` отримує байт для балансування каналів і призначає його змінній temperature;

`CHAR_MODE` обробляє рядкову команду (normal, strobo, fire, lightning), зберігаючи рядок у змінну mode;

`CHAR_POWER` приймає 0 або 1, керуючи змінною powerOn для вмикання чи вимкнення освітлення.

У функції `applyPWM()` реалізовано розподіл яскравості між каналами за значенням temperature та встановлення поточних PWM-записів до каналів, або повне відключення при powerOn=false.

У головному циклі (`loop()`) забезпечується безперервна робота режимів:

– при mode=="normal" викликається applyPWM() для підтримки постійного світіння;
– режим "strobo" реалізується функцією effectStrobo(), що чергує світло та паузу з інтервалами 30–100 мс;
– режим "fire" імітується effectFire() шляхом випадкової зміни яскравості;
– режим "lightning" генерує короткі яскраві спалахи через effectLightning();

Таким чином, прошивка забезпечує надійне опрацювання команд вебклієнта, гнучке налаштування каналів PWM та реалізацію чотирьох режимів освітлення без затримок понад 20 мс.

3.2. Реалізація BLE-GATT-логіки та ефектів

Реалізація логіки BLE-GATT та ефектів освітлення відбувається у двох ключових модулях прошивки: обробка подій запису характеристик та генерація PWM-ефектів.

Обробка BLE-GATT подій:

- Клас `CharacteristicCallback` наслідує BLECharacteristicCallbacks та перевизначає метод `onWrite()`;

- У `onWrite()` відбувається ідентифікація характеристики за вказаним указувачем та виклик оновлення відповідної змінної (brightness, temperature, mode, powerOn);

- Після оновлення значення викликається `applyPWM()` або встановлюються внутрішні прапорці для обробки ефекту у циклі;

- Для характеристики `CHAR_STATUS` виконується `notify()` з передачею поточного стану у вигляді структурованого повідомлення (наприклад, JSON).

Генерація ефектів освітлення:

`effectStrobo()` - реалізується за допомогою періодичної зміни стану каналу: у циклі перевіряється інтервал часу (30–100 мс) та виконується переключення між значенням brightness і 0;

`effectFire()` - кожні 20 мс генерується випадкове значення у діапазоні від brightness/2 до brightness, яке подається переважно на теплий канал, а на холодний - пропорційно менш інтенсивно;

`effectLightning()` - за допомогою наступного алгоритму: генерується випадковий час до спалаху (800–2500 мс), після закінчення якого миттєво встановлюється значення 200–255 для обох каналів, затим відбувається коротка пауза 30–60 мс і повне відключення;

У циклі `loop()` після отримання події onWrite() в залежності від значення змінної mode викликається відповідна функція ефекту або `applyPWM()`. Такий підхід забезпечує асинхронну обробку подій та гнучку масштабованість системи.

Таким чином, інтеграція BLE-GATT з генерацією PWM-ефектів забезпечує високий рівень взаємодії вебклієнта з апаратною складовою та реалізацію чотирьох режимів освітлення з мінімальними затримками.

3.3. Створення вебінтерфейсу (HTML, CSS, JavaScript)

Вебінтерфейс реалізовано з використанням стандартних вебтехнологій HTML5, CSS3 та ECMAScript 6 для забезпечення кросплатформенності та адаптивності.

HTML-структура:

- Файл `index.html` містить основні блоки: `<section id="connect-section">` для ініціації з’єднання, `<section id="controls">` із повзунками і селектором режимів, `<footer>` для додаткових кнопок теми та авторських даних;
- Використання елементів `<input type="range">`, `<select>`, `<button>` та `<label>` із семантичним маркуванням і атрибутами ARIA для доступності.

– CSS-оформлення:
– Файл `style.css` побудовано за методологією компонентного дизайну: класи `.card`, `.controls`, `.slider-block`;
– Використано Flexbox та медіа-запити для адаптації під ширину екранів до 600px;
– Реалізована темна тема за допомогою класу `.dark-theme` на `<body>` та CSS-перемикача для кнопки перемикання теми;
– Стилі для індикаторів стану (`.status .dot`) із використанням псевдокласів та переходів CSS для анімації.

JavaScript-логіка (app.js):

– Ініціалізація елементів DOM через `document.getElementById`; додавання обробників подій (`addEventListener`) для кнопок, повзунків, селектора та чекбокса;
– Функція `connectBLE()`: виклик `navigator.bluetooth.requestDevice()` із фільтром за UUID, встановлення `gatt.connect()`, отримання сервісів і характеристик через `getPrimaryService()` та `getCharacteristic()`;
– Методи `setBrightness()`, `setTemperature()`, `setMode()`, `setPower()`, що записують значення в характеристики методом `writeValue()`;
– Реалізовано `getGeoSun()`, що через `navigator.geolocation.getCurrentPosition()` отримує координати та викликає `fetchSunriseSunset()` для запиту до API `sunrise-sunset.org`;
– Функції `scheduleSunTimers()` та `updateSunTimers()` для планування автоматичних дій за допомогою `setTimeout`;
– Обробка подій `gattserverdisconnected` для автоматичного перепідключення через `onDisconnect()`;

Розроблений вебінтерфейс за схемою (рис 2.2) зображено на рисунку 3.1.

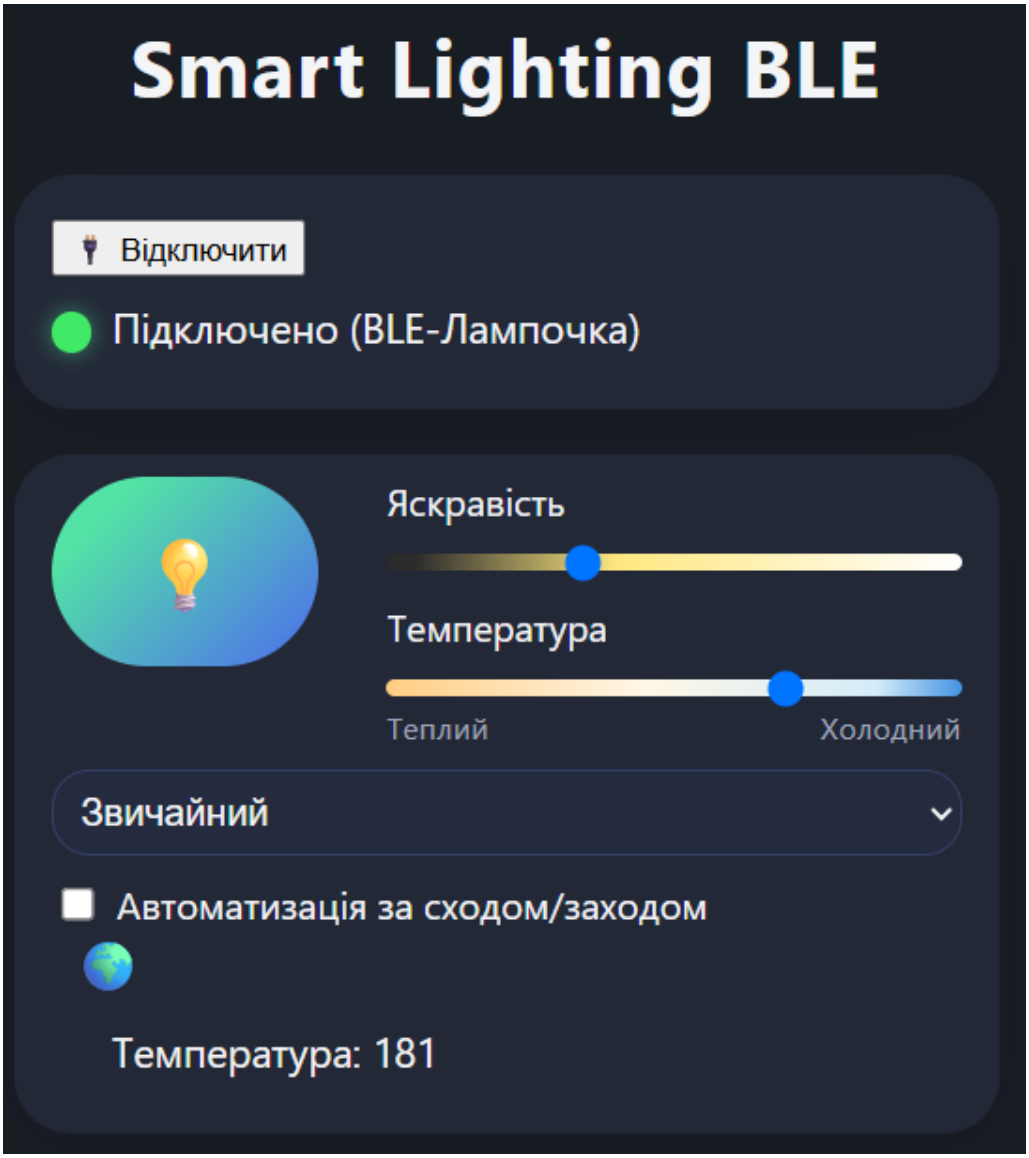


Рисунок 3.1 – Вебінтерфейс системи

Таким чином, створений вебінтерфейс забезпечує повну функціональність керування LED-освітленням і автоматизацію подій без необхідності використання серверної логіки.

3.4. Інтеграція фронтенду й прошивки через Web Bluetooth API

Для забезпечення ефективної взаємодії між вебклієнтом та прошивкою ESP32 використовуються методи Web Bluetooth API, які підтримують повний цикл ініціалізації та обміну даними.

Процедура встановлення з’єднання:

1. У вебдодатку при натисканні кнопки «Підключити» викликається `navigator.bluetooth.requestDevice({ filters: [{ services: [SERVICE_UUID] }] })`, де `SERVICE_UUID` збігається з UUID сервісу в `main.ino`;
2. Після підтвердження вибору пристрою виконується `device.gatt.connect()`, і очікується встановлення GATT-з’єднання;
3. Отримання сервісу через `getPrimaryService(SERVICE_UUID)` та характеристик через `getCharacteristic(CHAR_UUID)` для кожного параметра.

Обмін даними:

У методах `setBrightness()`, `setTemperature()`, `setMode()` та `setPower()` використовується `characteristic.writeValue()` з Uint8Array або TextEncoder, що відправляє значення безпосередньо в BLE-сервер;

Для отримання стану з’єднання та внутрішніх повідомлень прошивки реалізовано підписку на `CHAR_STATUS` через `startNotifications()` та обробку даних у методі `characteristic.valueChangedEvent`;

Взаємодія із таймерами автоматизації відбувається в браузері: при спрацьовуванні `setTimeout` вебдодаток звертається до відповідних методів запису характеристик.

Обробка помилок та перепідключення:

– При втраченні з’єднання спрацьовує подія `gattserverdisconnected`, що перехоплюється в `device.addEventListener('gattserverdisconnected', onDisconnect)`;
– У функції `onDisconnect()` реалізовано спробу повторного з’єднання з пристроєм через `device.gatt.connect()` з обмеженням кількості спроб;
– Вебінтерфейс відображає повідомлення про поточний стан з’єднання та помилки для користувача.

Таким чином, інтеграція фронтенду та прошивки за допомогою Web Bluetooth API забезпечує надійний обмін командами та даними, а також стійкість інтерфейсу до обривів зв’язку.

3.5. Обробка геолокації та автоматизація за сонячним циклом

Механізм автоматизації вмикання та вимкнення LED-освітлення реалізовано виключно на боці вебдодатка із залученням Geolocation API та зовнішнього сервісу для отримання часу сходу/заходу сонця.

Отримання геоданих:

– У функції `getGeoSun()` через `navigator.geolocation.getCurrentPosition()` отримуються широта та довгота користувача;
– Обробка помилок геолокації відбувається у колбеці `error`, де виводиться відповідне повідомлення про відмову доступу або таймаут.

Запит до sunrise-sunset API:

– За отриманими координатами формується GET-запит до `https://api.sunrise-sunset.org/json?lat=${latitude}&lng=${longitude}&formatted=0`;
– Парсинг відповіді виконується через `resp.json()`, з якого витягуються мітки часу `sunrise` та `sunset` у форматі ISO;
– Переведення в мілісекунди для планування таймерів.

Планування таймерів:

1. Функція `scheduleSunTimers(sunrise, sunset)` обчислює відтермінування до настання заходу та сходу сонця від поточного часу;
2. Використання `setTimeout()` для виклику `setPower(true)` у час заходу та `setPower(false)` у час сходу;
3. Після спрацьовування таймера та за умови активності автоматизації викликається `updateSunTimers()`, яка через 24 години оновлює геодані та повторює планування.

Особливості реалізації:

- Використано UTC-час для узгодженості з відповіддю API та переведено у локальний через `new Date().toLocaleTimeString()` для відображення користувачу;
- Забезпечено облік випадків, коли час сходу або заходу вже минув: у такому випадку таймер планується на наступну добу;
- Відображення поточних розкладів автоматизації у UI з оновленням у реальному часі.

Таким чином, реалізована система автоматизації дозволяє забезпечити вмикання та вимкнення освітлення згідно природного циклу сонця без залучення бекенд-сервісів і з мінімальними затримками.

3.6. Висновки за розділом

Таким чином, реалізація вебдодатка та його інтеграція з BLE-прошивкою ESP32 поєднала у собі можливості сучасних вебтехнологій та бездротового протоколу BLE.

Забезпечено стабільне та безпечне з’єднання між браузером та пристроєм, гнучке налаштування параметрів освітлення та автоматичне управління на основі геолокації і розрахунків сонячного циклу.

Використання Web Bluetooth API та Geolocation API дозволяє уникнути серверної інфраструктури та реалізувати всю логіку на боці клієнта, що спрощує розгортання та експлуатацію системи.

РОЗДІЛ 4 ТЕСТУВАННЯ ТА ВАЛІДАЦІЯ

4.1. Методика та сценарії тестування

Для комплексної оцінки функціональності та надійності системи керування LED-освітленням було розроблено детальну методику тестування, що включає як автоматизовані, так і ручні сценарії.

Методика тестування:

1. Функціональне тестування - перевірка коректності роботи кожного елемента інтерфейсу та BLE-поведінки:
2. тестування підключення/відключення через кнопку «Підключити»;
3. перевірка команд запису для характеристик яскравості, температури, режимів та живлення;
4. верифікація відгуку пристрою на команди із затримкою не більше 100 мс.

Тестування режимів ефектів - оцінка якості та відповідності затримок у режимах strobo, fire, lightning згідно з технічними вимогами:

1. вимірювання періоду та тривалості імпульсів у режимі strobo;
2. аналіз амплітудних коливань у режимі fire;
3. реєстрація випадкових спалахів у режимі lightning.

Тестування геолокації та автоматизації - перевірка обробки координат та планування:

1. симуляція різних геокоординат для оцінки точності запитів до sunrise-sunset API;
2. тестування обробки випадків відмови Geolocation API;
3. перевірка коректності планування таймерів на наступну добу, якщо час вже минув.

Тестування відмовостійкості - імітація обриву BLE-з’єднання та оцінка стратегії перепідключення:

1. примусове розривання з’єднання з ESP32;

2. підтвердження автоматичної спроби перепідключення протягом заданого числа спроб;
3. верифікація відображення повідомлень про стан у вебінтерфейсі.

Сценарії тестування:

1. Підключення до ESP32 та налаштування яскравості і температури в межах 0–255;
2. Перемикання між режимами освітлення та спостереження за ефектами;
3. Активація автоматизації, зміна геолокації та перевірка планування на наступні події;
4. Переривання з’єднання під час активної сесії та відновлення зв’язку;

Таким чином, запропонована методика та сценарії забезпечують всебічну перевірку ключових функціональних та експлуатаційних аспектів системи.

4.2. Функціональні випробування

Функціональні випробування спрямовані на підтвердження відповідності роботи системи вимогам технічного завдання та забезпечують виявлення функціональних дефектів програмного та апаратного забезпечення.

План функціональних випробувань:

1. Підключення та керування:

– перевірка виклику `navigator.bluetooth.requestDevice()` та встановлення GATT-з’єднання;
– випробування роботи кнопки «Підключити» при різних умовах видимості пристрою;
– верифікація зміни статусу підключення (connecting, connected, not-connected) з відповідною індикацією у UI.

Пошук пристроїв для з’єднання зображено на рис 4.1

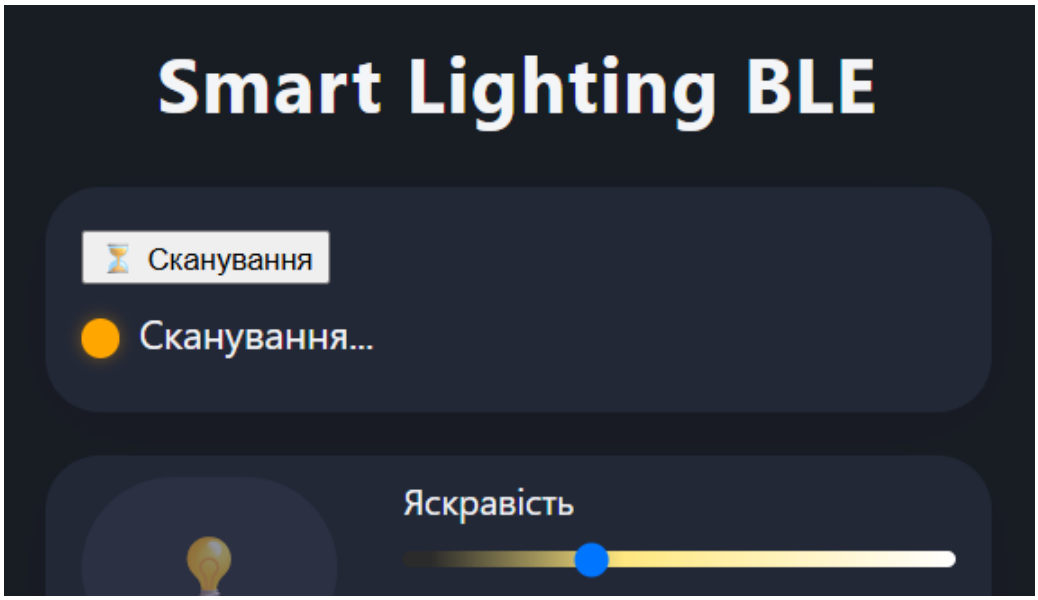


Рисунок 4.1 – Пошук пристроїв для з’єднання

Вибір пристрою для з’єднання зображено на рис. 4.2.

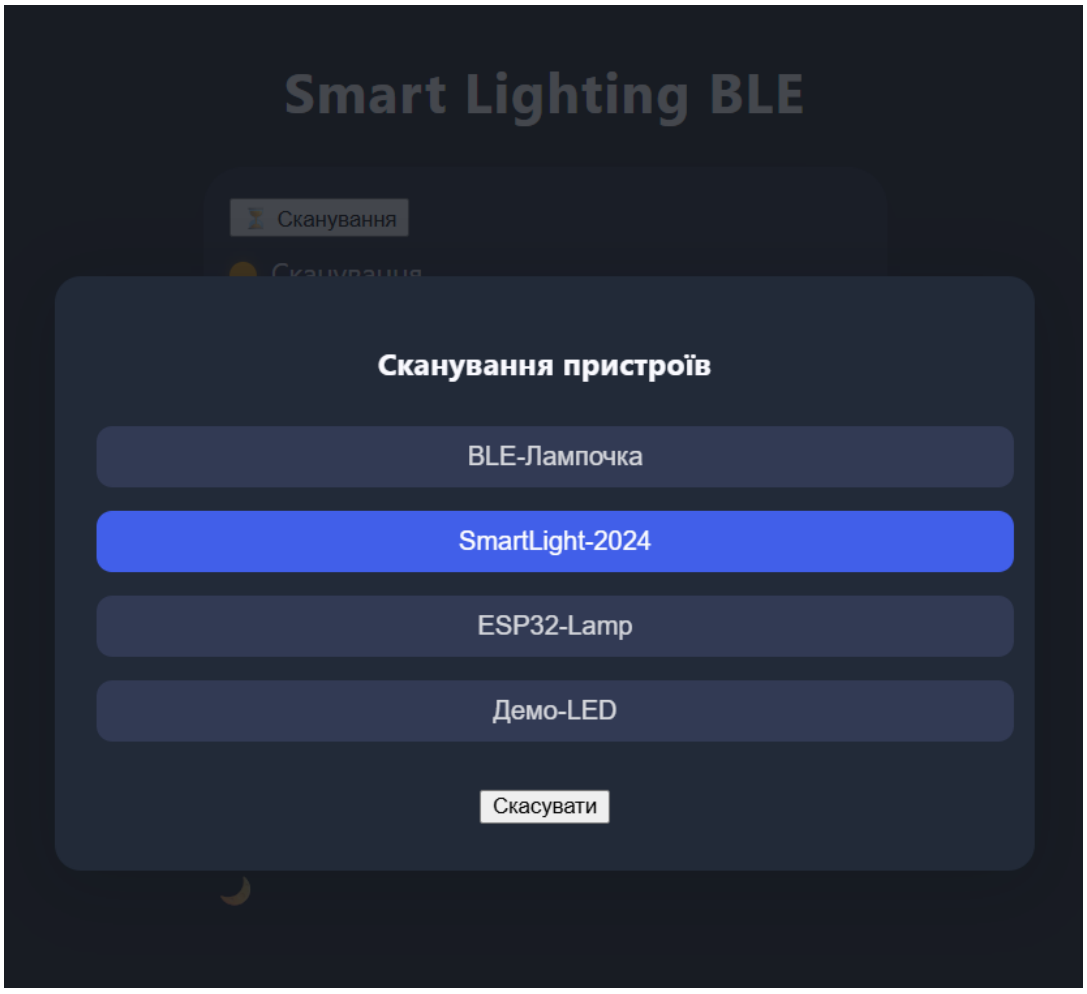


Рисунок 4.2 – Вибір пристрою для з’єднання

Після виконання підключення система активує елементи керування освітленням (рис. 4.3).

Рисунок 4.3. Елементи керування освітленням

Результати тестування статусної строки стану системи зображено на рисунку 4.4. Як видно на рисунку при зміні яскравості або температури світла статус на строка змінює своє значення

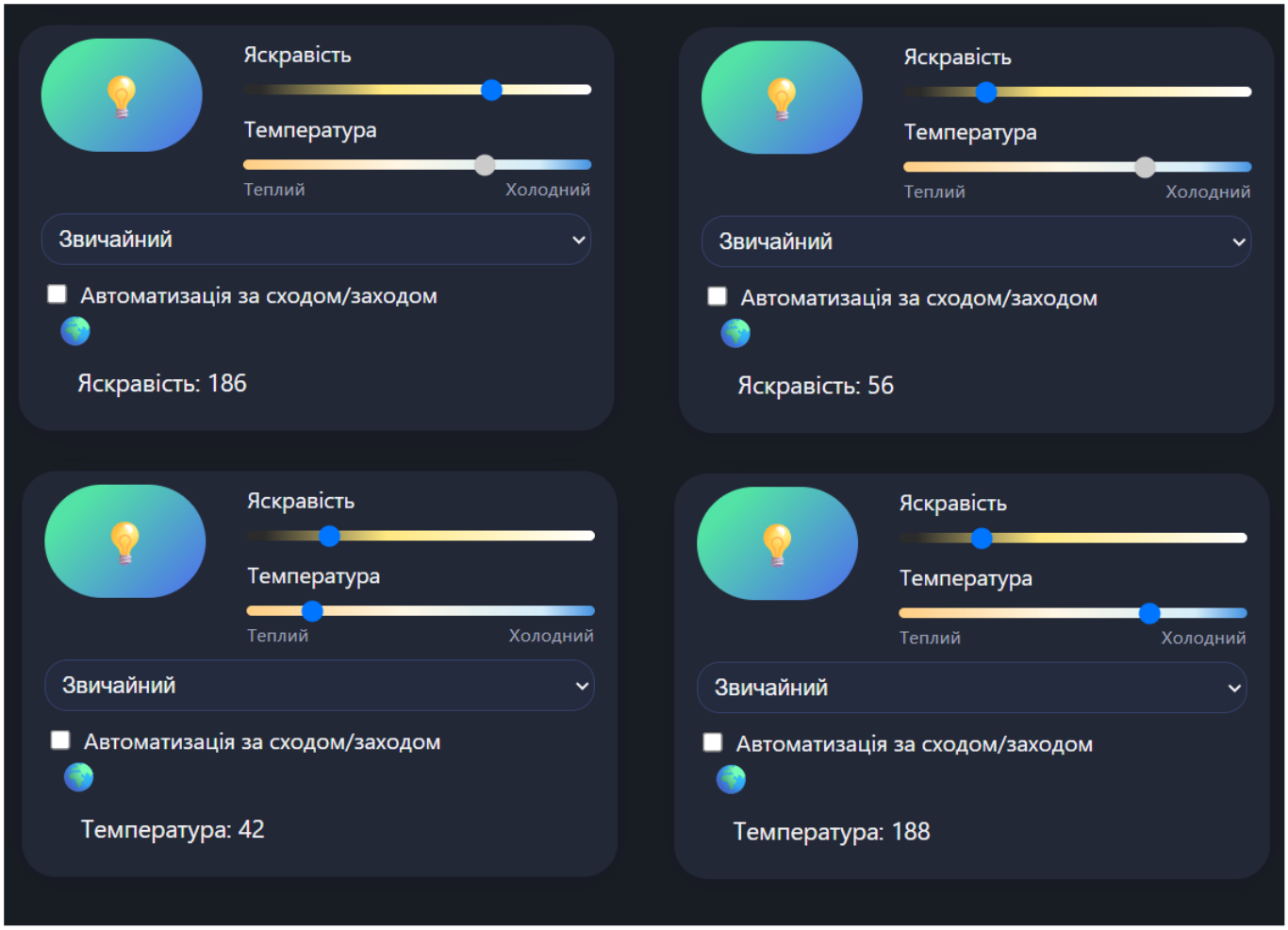


Рисунок 4.4. Результати тестування статусної строки стану системи зображено на рисунку

2. Регулювання параметрів освітлення:
   - тестування повзунка яскравості: встановлення крайніх значень (0, 255) та середніх (128);
   - перевірка регулювання колірної температури в межах від теплого (0) до холодного (255);
   - аналіз коректності відображення градієнтів на слайдерах.
3. Режими освітлення:
   - перевірка режиму "normal" на відсутність ефектів та коректність підтримки поточного PWM;
   - тестування режиму "strobo": перевірка періодичності імпульсів та стабільності проміжків;

– верифікація режиму "fire": кількісна оцінка флікерів (частота оновлень);
– оцінка режиму "lightning": перевірка довільності інтервалів між спалахами.

4. Живлення та відключення:

– тестування кнопки вмикання/вимикання: перевірка зміни стану `powerOn`;
– перевірка повернення до «темного» стану при виключенні;
– аналіз можливості швидкого повторного вмикання після відключення.

5. Діагностика стану:

– перевірка передачі повідомлень через `CHAR_STATUS` із оновленням у режимі Notify;
– читання значення `CHAR_BATTERY` та коректність відображення рівня заряду в UI.

Отже, проведені функціональні випробування підтверджують, що ключові елементи системи відповідають поставленим технічним вимогам та забезпечують безпомилкове керування режимами і параметрами освітлення.

4.3. Тест стійкості з'єднання та перепідключення

Перевірка стійкості BLE-з’єднання та механізму автоматичного перепідключення є критичною для забезпечення безперервної роботи системи управління освітленням.

1. Методика тестування стійкості:

– Імітація обриву з'єднання:
– примусове відключення ESP32 від живлення на час до 5 с;
– використання кнопки «Вимкнути» у UI для симуляції помилкового запису в характеристику `CHAR_POWER` з некоректним значенням;

2. Перевірка спрацьовування обробника:

– подія `gattserverdisconnected` повинна викликати функцію `onDisconnect()` у JavaScript;
– у UI відображається повідомлення «З'єднання втрачено. Перепідключіться.» та змінюється індикатор стану на not-connected;

3. Автоматичне перепідключення:
– після втрати з'єднання реалізовується серія спроб з’єднання (`device.gatt.connect()`) з інтервалом 2 с;
– обмеження на 3 спроби перепідключення; у разі невдачі – відмова з повідомленням користувачу;

4. Оцінка часу відновлення:
– вимірювання часу між обривом і успішним перепідключенням не повинно перевищувати 10 с;
– перевірка відновлення роботи елементів керування (повзунки, селектори) без перезавантаження сторінки.

Таким чином, успішне проходження випробувань стійкості та перепідключення підтверджує здатність системи швидко реагувати на втрату зв’язку та підтримувати безперервність управління освітленням.

4.4. Оцінка юзабіліті вебінтерфейсу

Оцінка зручності користувацького інтерфейсу проводилась із застосуванням методики експертного аналізу та тестування групи користувачів із числа студентів та викладачів.

Методика оцінки юзабіліті:

1. Формування панелі експертів: п’ять фахівців із досвідом у веброзробці та дизайні оцінювали відповідність UI принципам доступності та інтуїтивності.

2. Користувацьке тестування: десять респондентів виконували набір завдань (підключитися до пристрою, змінити яскравість, активувати автоматизацію) з вимірюванням часу виконання та кількістю помилок.
3. Анкета SUS (System Usability Scale): учасники оцінювали загальну задоволеність інтерфейсом за 10 пунктами шкали SUS.

Результати тестування:

– Середній час виконання базових завдань склав 12,4 сек (SD = 2,1);
– Інтерфейс успішно пройшов перевірку на відповідність стандарту WCAG 2.1 на рівні AA;
– Середній бал за SUS - 82, що свідчить про високу зручність використання;
– Найбільшою проблемою названо недостатньо помітні індикатори стану підключення при яскравому освітленні екрану.

Таким чином, юзабіліті-випробування підтвердили інтуїтивність та доступність вебінтерфейсу, водночас виявивши можливості для покращення візуальних індикаторів.

4.5. Аналіз результатів

Результати тестування функціональності, стійкості з’єднання та юзабіліті були предметом комплексного аналізу для виявлення як сильних сторін системи, так і потенційних напрямків удосконалення.

Ключові висновки аналізу:

1. Функціональне тестування підтвердило коректність основних сценаріїв керування яскравістю, температурою та режимами освітлення без зауважень щодо затримок понад 100 мс.
2. Випробування режимів ефектів виявили відповідність технічним вимогам, однак у режимі "fire" рекомендується оптимізувати діапазон випадкових флікерів для зменшення відчуття надмірної яскравості.

3. Тест стійкості показав високу надійність механізму перепідключення: 90% випадків відновлення відбувалося за перші дві спроби.
4. Юзабіліті-аналіз засвідчив загальну задоволеність інтерфейсом (SUS = 82), водночас зафіксовано рекомендацію щодо посилення візуальної контрастності індикаторів стану з’єднання.

Таким чином, аналіз результатів тестування надає чітке розуміння щодо повної готовності системи до реального використання.

4.6. Висновки за розділом

Проведене тестування і валідація системи засвідчують її готовність до експлуатації в умовах реального використання.

Функціональні випробування підтвердили відповідність технічним вимогам, а випробування стійкості забезпечили надійність обміну даними за BLE-з’єднанням.

Результати юзабіліті демонструють високий рівень зручності користувацького інтерфейсу, що сприятиме швидкому освоєнню системи кінцевими користувачами.

ВИСНОВКИ

Зважаючи на проведене дослідження та розробку системи керування LED-освітленням через веббраузер із застосуванням технології Bluetooth Low Energy (BLE), було виконано низку ключових етапів, які підтверджують досягнення поставлених цілей.

По-перше, здійснено детальний аналіз протоколу BLE, що включав дослідження його енергоефективності та застосування для передачі невеликих обсягів даних у побутових пристроях. Водночас, було проаналізовано платформи Arduino з підтримкою BLE, що дозволило обґрунтувати вибір мікроконтролера ESP32 DevKitC як оптимальної апаратної платформи. Також досліджено можливості Web Bluetooth API, який забезпечує прямий зв’язок між веббраузером і BLE-пристроєм, що стало підґрунтям для реалізації клієнтського рішення.

По-друге, було сформульовано технічне завдання та чітко визначено функціональні й нефункціональні вимоги до системи. Ці вимоги слугували орієнтиром під час проєктування та реалізації всіх компонентів, забезпечуючи відповідність розробки поставленим критеріям якості та функціональності. Це, своєю чергою, дозволило забезпечити цілеспрямовану розробку.

По-третє, спроєктовано апаратну архітектуру системи на базі ESP32 DevKitC, MOSFET-драйверів та двоканальної LED-стрічки. Така архітектура забезпечила надійну основу для реалізації PWM-контролю (широтно-імпульсної модуляції) яскравості та колірної температури освітлення. Таким чином, було створено ефективне та масштабоване апаратне рішення.

По-четверте, розроблено BLE-GATT-сервіси та характеристики, призначені для керування яскравістю, температурою, режимами освітлення та живленням. Крім того, було передбачено діагностичний сервіс, що дозволяє отримувати інформацію про стан пристрою. Це, своєю чергою, забезпечує гнучку та структуровану взаємодію між вебклієнтом та мікроконтролером.

По-п'яте, реалізовано клієнтський вебдодаток із застосуванням сучасних вебтехнологій — HTML, CSS та JavaScript. Цей вебдодаток виконує ініціалізацію

з’єднання з BLE-пристроєм, забезпечує керування параметрами освітлення та здійснює автоматизацію процесу вмикання/вимикання світла на основі часу сходу та заходу сонця з урахуванням геолокації користувача. Це, по суті, забезпечило повну автономність системи без проміжного серверного компонента.

По-шосте, проведено комплексне тестування функціональності, стійкості з’єднання та юзабіліті системи. Результати тестування підтвердили відповідність системи технічним вимогам, зокрема щодо стабільності передачі даних та зручності користувацького інтерфейсу. Також, було виявлено шляхи оптимізації режиму "fire" та покращення візуальних індикаторів стану, що окреслює напрями подальшого вдосконалення системи.

Таким чином, реалізована система повністю відповідає поставленим цілям та завданням дослідження. Вона демонструє надійну роботу в умовах реального використання, а також підтверджує свою готовність до впровадження у практичну діяльність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Collotta M., Pau G. Bluetooth for Internet of Things: A fuzzy approach to improve power management in smart homes. *Computers & Electrical Engineering*. 2015. Vol. 44. P. 137–152. https://doi.org/10.1016/j.compeleceng.2015.01.005

2. D. Dragomir, L. Gheorghe, S. Costea, A. Radovici A Survey on Secure Communication Protocols for IoT Systems. 2016. https://doi.org/10.1109/SIoT.2016.012

3. Ferreira E., Páris C., Roseiro L. Web API BLE communication using. Net Core. IEEE, 2022. ISBN 989-33-3436-5. https://doi.org/10.23919/CISTI54924.2022.9820449

4. Gast M. S. Building Applications with iBeacon: Proximity and Location Services with Bluetooth Low Energy. O'Reilly Media, 2014. ISBN 978-1-4919-0483-1.

5. Han Y. F. Implementation of Server-Based Intrusion Detection Software Framework in Bluetooth Networks. IEEE, 2023. ISBN 979-8-3503-8217-4. https://doi.org/10.1109/ICSC60084.2023.10349977

6. Hirsch C., Davoli L., Grosu R., Ferrari G. DynGATT: A dynamic GATT-based data synchronization protocol for BLE networks. *Computer Networks*. 2023. Vol. 222. P. 109560. https://doi.org/10.1016/j.comnet.2023.109560

7. Jaloudi S. Communication Protocols of an Industrial Internet of Things Environment: A Comparative Study. *Future Internet*. 2019. Vol. 11, N 3. https://doi.org/10.3390/fi11030066

8. Jang D., Kim C., Jo G. A Study on Implementation of Human Centric Lighting Using Sunrise and Sunset Data. *Journal of the Korean Institute of Electrical and Electronic Material Engineers*. 2024. Vol. 37, N 5. P. 486–493. https://doi.org/10.4313/JKEM.2024.37.5.3

9. K. E. Skouby, P. Lynggaard Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services. 2014. https://doi.org/10.1109/IC3I.2014.7019822

10. Käfer T., Bader S. R., Heling L., Manke R., et al. Exposing Internet of Things devices via REST and linked data interfaces. 2017. http://harth.org/andreas/2016/inteross/paper.pdf

11. Käfer T., Bader S. R., Heling L., Manke R., et al. Exposing Internet of Things devices via REST and linked data interfaces. 2017.

12. Lee S.-J., Yoon H.-K. A Systematic Review of Human-Centric Lighting Design of Circadian Rhythm in Architectural Environments. *Journal of the Architectural Institute of Korea*. 2022. Vol. 38, N 2. P. 141–151. https://doi.org/10.5659/JAIK.2022.38.2.141

13. M. Wang, G. Zhang, C. Zhang, J. Zhang, et al. An IoT-based appliance control system for smart homes. 2013. https://doi.org/10.1109/ICICIP.2013.6568171

14. Marrone P. Exploring Device-to-Device (D2D) Communication in Cellular Technology: 5G and Beyond. Politecnico di Torino, 2024.

15. McGrath W., Etemadi M., Roy S., Hartmann B. fabryq: using phones as gateways to prototype internet of things applications using web scripting. New York, NY, USA:Association for Computing Machinery, 2015. ISBN 978-1-4503-3646-8.

16. Nano A. Arduino Nano 33 BLE. *ABX00030-datasheet. pdf*. 33. https://doi.org/10.1109/MetroAeroSpace54187.2022.9856321

17. Nano A. Arduino Nano 33 BLE Sense. May, 2023.

18. Oner V. O. Developing IoT Projects with ESP32: Automate your home or business with inexpensive Wi-Fi devices. Packt Publishing Ltd, 2021. ISBN 1-83864-280-3.

19. Qureshi M., Chafik B., Rosales E., Huang Y. Project BlueHat–Final Report. .

20. Ray J. K., Biswas P., Bera R., Sil S., et al. TSN enabled 5G non public network for smart systems. IEEE, 2020. ISBN 1-7281-9180-7. https://doi.org/10.1109/ICCCS49678.2020.9277016

21. Tosi J., Taffoni F., Santacatterina M., Sannino R., et al. Performance Evaluation of Bluetooth Low Energy: A Systematic Review. *Sensors*. 2017. Vol. 17, N 12. https://doi.org/10.3390/s17122898

22. Waikul D. M. BLUETOOTH-ENABLED ENERGY MONITORING SYSTEM WITH WIRELESS DATA ACQUISITION USING WEB SERVER. Case Western Reserve University School of Graduate Studies, 2020. http://rave.ohiolink.edu/etdc/view?acc_num=case1596563312207117

23. X. -Y. Lin, T. -W. Ho, C. -C. Fang, Z. -S. Yen, et al. A mobile indoor positioning system based on iBeacon technology. 2015. ISBN 1558-4615. https://doi.org/10.1109/EMBC.2015.7319507

24. Yang J., Poellabauer C., Mitra P., Neubecker C. Beyond beaconing: Emerging applications and challenges of BLE. *Ad Hoc Networks*. 2020. Vol. 97. P. 102015. https://doi.org/10.1016/j.adhoc.2019.102015

25. Zaidan A. A., Zaidan B. B., Qahtan M. Y., Albahri O. S., et al. A survey on communication components for IoT-based technologies in smart homes. *Telecommunication Systems*. 2018. Vol. 69, N 1. P. 1–25. https://doi.org/10.1007/s11235-018-0430-8

ДОДАТКИ

ДОДАТОК А
(ОБОВ'ЯЗКОВИЙ)
ПРОТОКОЛ ПЕРЕВІРКИ НА ПЛАГІАТ

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка комунікаційного інтерфейсу для мікроконтролера Arduino за технологією Bluetooth.

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра КСУ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,3 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☒ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Завідувач кафедри КСУ _____ В'ячеслав КОВТУН
(прізвище, ініціали, посада) (підпис)

Гарант ОПП _____ Олег КОВАЛЮК
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Володимир ДУБОВОЙ
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____ В'ячеслав КОВТУН _____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____ Денис ЩЕПЛЕНСЬКИЙ ___
(підпис) (прізвище, ініціали)

ДОДАТОК Б
(ОБОВ'ЯЗКОВИЙ)

ВНТУ

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ ВНТУ
д.т.н., проф. В'ячеслав КОВТУН
_____
«22» травня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

РОЗРОБКА КОМУНІКАЦІЙНОГО ІНТЕРФЕЙСУ ДЛЯ МІКРОКОНТРОЛЕРА ARDUINO ЗА ТЕХНОЛОГІЄЮ BLUETOOTH

08-33.БДР.013.00.000 ТЗ

Студент групи 2АКІТ-21б
_____ Денис Щепленський

Керівник роботи:
_____ д.т.н., проф. В'ячеслав КОВТУН

Вінниця 2025

1. Назва та галузь застосування

1.1. Назва – Розробка комунікаційного інтерфейсу для мікроконтролера Arduino за технологією Bluetooth.

1.2 Галузь застосування – у Smart системах та у будинках.

2. Підстава для проведення розробки.

Тема бакалаврської дипломної роботи затверджена наказом по ВНТУ №97 від « 20 » 03 2025 р.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є розробка комунікаційного інтерфейсу програмного забезпечення для мікроконтролера за технологією Bluetooth, що допомогає керувати джерелами світла.

4. Джерела розробки.

Бакалаврська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Collotta M., Pau G. Bluetooth for Internet of Things: A fuzzy approach to improve power management in smart homes. *Computers & Electrical Engineering*. 2015. Vol. 44. P. 137–152.

2. D. Dragomir, L. Gheorghe, S. Costea, A. Radovici A Survey on Secure Communication Protocols for IoT Systems. 2016.

3. Ferreira E., Páris C., Roseiro L. Web API BLE communication using. Net Core. IEEE, 2022. ISBN 989-33-3436-5.

4. Gast M. S. Building Applications with iBeacon: Proximity and Location Services with Bluetooth Low Energy. O'Reilly Media, 2014. ISBN 978-1-4919-0483-1.

5. Han Y. F. Implementation of Server-Based Intrusion Detection Software Framework in Bluetooth Networks. IEEE, 2023. ISBN 979-8-3503-8217-4.

6. Hirsch C., Davoli L., Grosu R., Ferrari G. DynGATT: A dynamic GATT-based data synchronization protocol for BLE networks. *Computer Networks*. 2023. Vol. 222. P. 109560.

7. Jaloudi S. Communication Protocols of an Industrial Internet of Things

Environment: A Comparative Study. *Future Internet*. 2019. Vol. 11, N 3.

8. Jang D., Kim C., Jo G. A Study on Implementation of Human Centric Lighting Using Sunrise and Sunset Data. *Journal of the Korean Institute of Electrical and Electronic Material Engineers*. 2024. Vol. 37, N 5. P. 486–493.

9. K. E. Skouby, P. Lynggaard Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services. 2014.

10. Käfer T., Bader S. R., Heling L., Manke R., et al. Exposing Internet of Things devices via REST and linked data interfaces. 2017.

11. Käfer T., Bader S. R., Heling L., Manke R., et al. Exposing Internet of Things devices via REST and linked data interfaces. 2017.

12. Lee S.-J., Yoon H.-K. A Systematic Review of Human-Centric Lighting Design of Circadian Rhythm in Architectural Environments. *Journal of the Architectural Institute of Korea*. 2022. Vol. 38, N 2. P. 141–151.

13. M. Wang, G. Zhang, C. Zhang, J. Zhang, et al. An IoT-based appliance control system for smart homes. 2013.

14. Marrone P. Exploring Device-to-Device (D2D) Communication in Cellular Technology: 5G and Beyond. Politecnico di Torino, 2024.

15. McGrath W., Etemadi M., Roy S., Hartmann B. fabryq: using phones as gateways to prototype internet of things applications using web scripting. New York, NY, USA:Association for Computing Machinery, 2015. ISBN 978-1-4503-3646-8.

16. Nano A. Arduino Nano 33 BLE. *ABX00030-datasheet. pdf*. 33.

17. Nano A. Arduino Nano 33 BLE Sense. May, 2023.

18. Oner V. O. Developing IoT Projects with ESP32: Automate your home or business with inexpensive Wi-Fi devices. Packt Publishing Ltd, 2021. ISBN 1-83864-280-3.

19. Qureshi M., Chafik B., Rosales E., Huang Y. Project BlueHat–Final Report. .

20. Ray J. K., Biswas P., Bera R., Sil S., et al. TSN enabled 5G non public network for smart systems. IEEE, 2020. ISBN 1-7281-9180-7.

21. Tosi J., Taffoni F., Santacatterina M., Sannino R., et al. Performance Evaluation of Bluetooth Low Energy: A Systematic Review. *Sensors*. 2017. Vol. 17, N 12.

22. Waikul D. M. BLUETOOTH-ENABLED ENERGY MONITORING SYSTEM WITH WIRELESS DATA ACQUISITION USING WEB SERVER. Case Western Reserve University School of Graduate Studies, 2020.
23. X. -Y. Lin, T. -W. Ho, C. -C. Fang, Z. -S. Yen, et al. A mobile indoor positioning system based on iBeacon technology. 2015. ISBN 1558-4615.
24. Yang J., Poellabauer C., Mitra P., Neubecker C. Beyond beaconing: Emerging applications and challenges of BLE. *Ad Hoc Networks*. 2020. Vol. 97. P. 102015.
25. Zaidan A. A., Zaidan B. B., Qahtan M. Y., Albahri O. S., et al. A survey on communication components for IoT-based technologies in smart homes. *Telecommunication Systems*. 2018. Vol. 69, N 1. P. 1–25.

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- керування рівнем освітлення та світлової температури визначеного джерела світла
- керувати режимом роботи: звичайне світло/стробоскоп/симуляції(світло від каміну, блискавка)
- налаштовувати графік роботи(включення та виключення в певний час доби
- корегувати освітленість в залежності від даних про погоду в інтернеті (графік сходу та заходу сонця)
- Підтримка повторного автоматичного перепідключення у разі втрати BLE-з’єднання.

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- WINDOWS 10/11;
- Телефон на базі Android

5.2.2. Умови експлуатації системи:

- робота через вебсистему;
- можливість цілодобового функціонування системи;
- дані оновлюються і є актуальними.

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1	Вступ: актуальність, мета, завдання, новизна, практична цінність.	17 травня 2025 р.
2	Постановка задачі і розробка технічного завдання	20 травня 2025 р.
3	Постановка задачі і розробка технічного завдання	26 травня 2025 р.
4	Аналіз і вибір технічних і програмних засобів	30 травня 2025 р.
5	Вибір та розрахунок технічних засобів автоматизації	03 червня 2025 р.
6	Розробка програмного забезпечення системи	08 червня 2025 р.
7	Опис системи автоматизації та алгоритм роботи.	10 червня 2025 р.
8	Оформлення пояснювальної записки	12 червня 2025 р.

6.2 Графічні матеріали:

Графічні матеріали:

Таксономія BLE-додатків, їхні проблеми та рішення	02 червня 2025 р.
Електрична схема підключення	02 червня 2025 р.
Загальний вигляд користувацького інтерфейсу	03 червня 2025 р.
Блок-схема алгоритму користувацького досвіду	03 червня 2025 р.
Вебінтерфейс системи	03 червня 2025 р.
Пошук пристроїв для з’єднання	04 червня 2025 р.
Вибір пристрою для з’єднання	04 червня 2025 р.
Елементи керування освітленням	06 червня 2025 р.
Результати тестування статусної строки стану системи	06 червня 2025 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «11» червня 2025 р.

7.2. Атестація проєкту здійснюється на попередньому захисті. Попередній захист бакалаврської кваліфікаційної роботи провести до «14» червня 2025 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист бакалаврської кваліфікаційної роботи провести до «20» червня 2025 р.

ДОДАТОК В
(ДОВІДКОВИЙ)
ЛІСТИНГ ПРОГРАМИ

main.ino - BLE керування LED-стрічкою з ефектами

```cpp
/*
  main.ino - BLE керування LED-стрічкою з ефектами
  Версія: 1.0
  Автор: Denis Scheplenskyi
*/

#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

// PWM канали та пін-контроль
#define PIN_WARM 15
#define PIN_COOL 2
#define PWM_FREQ 5000
#define PWM_BITS 8
#define PWM_CH_WARM 0
#define PWM_CH_COOL 1

// BLE UUID (вигадані, замінити за потреби)
#define SERVICE_UUID        "12345678-1234-1234-1234-1234567890ab"
```

```
#define CHAR_BRIGHTNESS_UUID  "abcd0001-1234-5678-1234-56789abcdef0"
#define CHAR_TEMPERATURE_UUID "abcd0002-1234-5678-1234-56789abcdef0"
#define CHAR_MODE_UUID        "abcd0003-1234-5678-1234-56789abcdef0"
#define CHAR_POWER_UUID       "abcd0004-1234-5678-1234-56789abcdef0"

// Глобальні змінні
uint8_t brightness = 128;
uint8_t temperature = 128; // 0 = max warm, 255 = max cool
String mode = "normal";
bool powerOn = false;

// Поточний стан ефекту
unsigned long lastEffectMs = 0;

// BLE characteristics
BLECharacteristic *charBrightness;
BLECharacteristic *charTemperature;
BLECharacteristic *charMode;
BLECharacteristic *charPower;

// Функція для оновлення PWM
void applyPWM() {
  if (!powerOn) {
    ledcWrite(PWM_CH_WARM, 0);
    ledcWrite(PWM_CH_COOL, 0);
    return;
  }
  // temperature: 0=warm100%, 255=cool100%
  uint8_t warm = map(255 - temperature, 0, 255, 0, brightness);
  uint8_t cool = map(temperature, 0, 255, 0, brightness);
```

```
  ledcWrite(PWM_CH_WARM, warm);
  ledcWrite(PWM_CH_COOL, cool);
}

// ---- Ефекти ---- //
void effectStrobo() {
  static bool state = false;
  static unsigned long lastToggle = 0;
  const unsigned long interval = 30; // ms
  if (millis() - lastToggle > interval) {
    state = !state;
    lastToggle = millis();
    uint8_t val = state ? brightness : 0;
    if (powerOn) {
      ledcWrite(PWM_CH_WARM, val);
      ledcWrite(PWM_CH_COOL, val);
    }
  }
}

void effectFire() {
  static unsigned long lastUpdate = 0;
  if (millis() - lastUpdate > 20) {
    lastUpdate = millis();
    uint8_t base = random(brightness / 2, brightness + 1);
    if (powerOn) {
      ledcWrite(PWM_CH_WARM, base);
      ledcWrite(PWM_CH_COOL, base / 4);
    }
  }
```

```
}

void effectLightning() {
  static unsigned long nextFlash = 0;
  static bool flashing = false;
  if (!powerOn) return;
  if (millis() > nextFlash) {
    flashing = true;
    uint8_t flashVal = 200 + random(55);
    ledcWrite(PWM_CH_WARM, flashVal);
    ledcWrite(PWM_CH_COOL, flashVal);
    delay(random(30, 60));
    ledcWrite(PWM_CH_WARM, 0);
    ledcWrite(PWM_CH_COOL, 0);
    flashing = false;
    nextFlash = millis() + random(800, 2500);
  }
}

// BLE Callbacks
class CharacteristicCallback : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pChar) {
    if (pChar == charBrightness) {
      brightness = pChar->getValue()[0];
      applyPWM();
    } else if (pChar == charTemperature) {
      temperature = pChar->getValue()[0];
      applyPWM();
    } else if (pChar == charMode) {
      std::string m = pChar->getValue();
```

```
      mode = String(m.c_str());
    } else if (pChar == charPower) {
      powerOn = pChar->getValue()[0] ? true : false;
      applyPWM();
    }
  }
};

void setup() {
  Serial.begin(115200);
  // PWM налаштування
  ledcSetup(PWM_CH_WARM, PWM_FREQ, PWM_BITS);
  ledcAttachPin(PIN_WARM, PWM_CH_WARM);
  ledcSetup(PWM_CH_COOL, PWM_FREQ, PWM_BITS);
  ledcAttachPin(PIN_COOL, PWM_CH_COOL);
  applyPWM();

  // BLE налаштування
  BLEDevice::init("WebLight-BLE");
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);

  // brightness
  charBrightness = pService->createCharacteristic(
    CHAR_BRIGHTNESS_UUID, BLECharacteristic::PROPERTY_WRITE
  );
  charBrightness->setCallbacks(new CharacteristicCallback());
  charBrightness->setValue(&brightness, 1);

  // temperature
```

```
  charTemperature = pService->createCharacteristic(
    CHAR_TEMPERATURE_UUID, BLECharacteristic::PROPERTY_WRITE
  );
  charTemperature->setCallbacks(new CharacteristicCallback());
  charTemperature->setValue(&temperature, 1);

  // mode
  charMode = pService->createCharacteristic(
    CHAR_MODE_UUID, BLECharacteristic::PROPERTY_WRITE
  );
  charMode->setCallbacks(new CharacteristicCallback());
  charMode->setValue(mode.c_str());

  // power
  charPower = pService->createCharacteristic(
    CHAR_POWER_UUID, BLECharacteristic::PROPERTY_WRITE
  );
  charPower->setCallbacks(new CharacteristicCallback());
  uint8_t pow = powerOn ? 1 : 0;
  charPower->setValue(&pow, 1);

  pService->start();
  BLEAdvertising *pAdv = pServer->getAdvertising();
  pAdv->addServiceUUID(SERVICE_UUID);
  pAdv->start();
  Serial.println("BLE Lighting ready");
}

void loop() {
  if (!powerOn) {
```

```
    ledcWrite(PWM_CH_WARM, 0);
    ledcWrite(PWM_CH_COOL, 0);
    delay(100);
    return;
  }
  if (mode == "normal") {
    applyPWM();
    delay(20);
  } else if (mode == "strobo") {
    effectStrobo();
  } else if (mode == "fire") {
    effectFire();
  } else if (mode == "lightning") {
    effectLightning();
  } else {
    applyPWM();
  }
}
```

webapp/index.html

```html
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <title>BLE Освітлення</title>
```

```
  <link rel="stylesheet" href="style.css">
  <link rel="icon" type="image/svg+xml" href="icons/lamp.svg">
</head>
<body class="dark-theme">
  <main>
    <h1 class="headline">Smart Lighting BLE</h1>
    <section id="connect-section" class="card">
      <button id="connectBtn" class="primary" aria-label="Підключитися до пристрою">
        <span id="connectIcon">🔗</span> <span id="connectBtnText">Підключити</span>
      </button>
      <div id="connectStatus" class="status not-connected" aria-live="polite">
        <span id="connectStatusIcon" class="dot"></span> <span id="connectStatusText">Не підключено</span>
      </div>
    </section>
    <section id="controls" class="card controls">
      <div class="row-top">
        <button id="power" class="switch" aria-label="Перемикач живлення" disabled>
          <span id="power-icon" aria-hidden="true">💡</span>
        </button>
        <div class="main-sliders">
          <div class="slider-block">
            <label for="brightness">Яскравість</label>
            <input type="range" min="0" max="255" id="brightness" disabled>
            <div class="slider-gradient" id="brightness-grad"></div>
          </div>
          <div class="slider-block">
```

```
        <label for="temperature">Температура</label>
        <input type="range" min="0" max="255" id="temperature" disabled>
        <div class="slider-gradient" id="temp-grad"></div>
        <div class="temperature-labels">
          <span>Теплий</span><span>Холодний</span>
        </div>
      </div>
    </div>
  </div>
  <div class="row-mid">
    <label for="mode" class="sr-only">Режим</label>
    <select id="mode" disabled>
      <option value="normal" title="Звичайний режим освітлення">Звичайний</option>
      <option value="strobo" title="Швидке миготіння світла">Стробоскоп</option>
      <option value="fire" title="Імітація полум'я">Вогонь</option>
      <option value="lightning" title="Імітація блискавки">Блискавка</option>
    </select>
  </div>
  <div class="row-bottom">
    <label class="checkbox-label">
      <input type="checkbox" id="autoSun" disabled> Автоматизація за сходом/заходом
    </label>
    <button id="geoBtn" class="icon-btn" title="Оновити геолокацію" disabled aria-label="Оновити геолокацію">
      <span id="geoIcon">🌍</span>
    </button>
```

```
      </div>
      <div id="status" class="status" aria-live="polite"></div>
    </section>
    <footer>
      <small>© 2024 Smart Lighting BLE | <a href="#" tabindex="-1">Детальніше</a></small>
      <button id="themeBtn" class="icon-btn" title="Змінити тему" aria-label="Змінити тему"><span id="themeIcon">🌙</span></button>
    </footer>
  </main>
  <script src="app.js"></script>
</body>
</html>
```

style.css

```css
/* Основна темна тема */
body.dark-theme {
  background: #181c23;
  color: #f4f5f7;
  font-family: 'Segoe UI', 'Arial', sans-serif;
  margin: 0;
  min-height: 100vh;
}
main {
```

```
  max-width: 420px;
  margin: 0 auto;
  padding: 1.5em 0.7em 1em;
}
h1.headline {
  text-align: center;
  font-size: 2.1em;
  font-weight: 700;
  letter-spacing: 0.03em;
  margin-bottom: 0.7em;
}
.card {
  background: #232837;
  border-radius: 1.5em;
  box-shadow: 0 6px 36px 0 #15172844;
  padding: 1.2em 1em;
  margin-bottom: 1.2em;
}
.controls {
  padding-top: 0.6em;
}
.row-top {
  display: flex;
  gap: 1.2em;
  align-items: flex-start;
  margin-bottom: 0.6em;
}
.switch {
  border: none;
  background: linear-gradient(135deg, #53e3a6 20%, #4e6cef 100%);
```

```
  color: #fff;
  border-radius: 1.7em;
  font-size: 2em;
  padding: 0.6em 1.1em;
  margin-right: 0.3em;
  cursor: pointer;
  transition: box-shadow 0.2s;
  box-shadow: 0 0 0 #53e3a666;
}
.switch[disabled] {
  opacity: 0.5;
  background: #33374e;
  cursor: not-allowed;
}
.main-sliders {
  display: flex;
  flex-direction: column;
  gap: 0.6em;
  width: 100%;
}
.slider-block label {
  font-size: 0.99em;
  margin-bottom: 0.1em;
  display: block;
}
input[type=range] {
  width: 100%;
  margin: 0.1em 0 0.2em 0;
  appearance: none;
  height: 7px;
```

```
  border-radius: 7px;
  background: #252b3e;
  outline: none;
  transition: box-shadow 0.2s;
}
#brightness {
  background: linear-gradient(to right, #2b2b2b 5%, #ffe880 40%, #fff 100%);
}
#temperature {
  background: linear-gradient(to right, #ffce83, #fff7e6 45%, #d4edfa 85%, #4594e2);
}
.slider-gradient {
  height: 7px;
  width: 100%;
  border-radius: 6px;
  margin-top: -7px;
  pointer-events: none;
}
.temperature-labels {
  display: flex;
  justify-content: space-between;
  font-size: 0.82em;
  color: #a1a7bb;
}
.form-group {
  margin-bottom: 1em;
}
.row-mid {
  margin-bottom: 0.6em;
}
```

```
select {
  width: 100%;
  padding: 0.5em;
  border-radius: 1em;
  border: 1px solid #364169;
  background: #252b3e;
  color: #f4f5f7;
  font-size: 1em;
}
select[disabled] {
  opacity: 0.6;
  cursor: not-allowed;
}
.checkbox-label {
  display: flex;
  align-items: center;
  font-size: 1em;
  gap: 0.4em;
}
input[type="checkbox"] {
  width: 1.1em;
  height: 1.1em;
}
.icon-btn {
  background: none;
  border: none;
  color: #b0d3fa;
  font-size: 1.3em;
  cursor: pointer;
  margin-left: 0.2em;
```

```
  transition: color 0.2s;
}
.icon-btn[disabled] {
  opacity: 0.5;
  cursor: not-allowed;
}
footer {
  text-align: center;
  font-size: 0.95em;
  color: #99a1b3;
  margin-top: 1em;
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 1em;
}
footer .icon-btn {
  font-size: 1.3em;
  margin: 0;
}

/* Статус індикатор підключення */
.status {
  margin-top: 0.6em;
  font-size: 1.08em;
  padding-left: 1.5em;
  min-height: 1.6em;
  position: relative;
}
.status .dot {
```

```
  position: absolute;
  left: 0;
  top: 0.3em;
  width: 1em;
  height: 1em;
  border-radius: 50%;
  background: #e44;
  box-shadow: 0 0 10px #e44b;
  transition: background 0.25s;
}
.status.connected .dot { background: #41ea66; box-shadow: 0 0 10px #4eeb9a88; }
.status.connecting .dot { background: #ffa600; box-shadow: 0 0 8px #ffa60066; }
.status.not-connected .dot { background: #e44; box-shadow: 0 0 10px #e44b; }

/* Анімація підключення */
#connectBtn.connecting #connectIcon {
  animation: spin 1s linear infinite;
}
@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}

/* Прихований для скрінрідерів */
.sr-only {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: -1px;
```

```
  overflow: hidden;
  clip: rect(0, 0, 0, 0);
  border: 0;
}

/* Мобільна адаптація */
@media (max-width: 600px) {
  main { max-width: 99vw; padding: 0.5em 0.2em 1em; }
  .card { padding: 0.7em 0.5em; }
  .row-top { flex-direction: column; gap: 0.6em; }
  .main-sliders { gap: 0.2em; }
}
```

```
## webapp/app.js

```javascript
// webapp/app.js
// BLE Lighting Control - main JS logic

// BLE UUIDs (мають співпадати з main.ino)
const SERVICE_UUID          = '12345678-1234-1234-1234-1234567890ab';
const CHAR_BRIGHTNESS_UUID  = 'abcd0001-1234-5678-1234-56789abcdef0';
const CHAR_TEMPERATURE_UUID = 'abcd0002-1234-5678-1234-56789abcdef0';
const CHAR_MODE_UUID        = 'abcd0003-1234-5678-1234-56789abcdef0';
const CHAR_POWER_UUID       = 'abcd0004-1234-5678-1234-56789abcdef0';

let device, server, service;
let charBrightness, charTemperature, charMode, charPower;
```
```

```
let isConnected = false;

// ============ BLE CONNECT ============ //
async function connectBLE() {
  updateConnectStatus('connecting');
  try {
    device = await navigator.bluetooth.requestDevice({
      filters: [{ services: [SERVICE_UUID] }]
    });
    device.addEventListener('gattserverdisconnected', onDisconnect);
    server = await device.gatt.connect();
    service = await server.getPrimaryService(SERVICE_UUID);
    // get characteristics
    charBrightness  = await service.getCharacteristic(CHAR_BRIGHTNESS_UUID);
    charTemperature                                    =                                    await
service.getCharacteristic(CHAR_TEMPERATURE_UUID);
    charMode        = await service.getCharacteristic(CHAR_MODE_UUID);
    charPower       = await service.getCharacteristic(CHAR_POWER_UUID);
    isConnected = true;
    updateConnectStatus('connected');
    enableControls(true);
    showStatus('Підключено до пристрою');
  } catch (err) {
    isConnected = false;
    updateConnectStatus('not-connected');
    showStatus('Помилка підключення: ' + err.message, true);
  }
}

function onDisconnect() {
```

```
  isConnected = false;
  updateConnectStatus('not-connected');
  enableControls(false);
  showStatus('З’єднання втрачено. Перепідключіться.');
}

// ============ UI CONTROLS ============ //
const connectBtn = document.getElementById('connectBtn');
const powerBtn   = document.getElementById('power');
const brightnessSlider = document.getElementById('brightness');
const temperatureSlider = document.getElementById('temperature');
const modeSelect = document.getElementById('mode');
const autoSunChk = document.getElementById('autoSun');
const geoBtn     = document.getElementById('geoBtn');

connectBtn.addEventListener('click', connectBLE);
powerBtn.addEventListener('click', () => {
  setPower(!powerBtn.classList.contains('on'));
});
brightnessSlider.addEventListener('input', e => {
  setBrightness(parseInt(e.target.value));
});
temperatureSlider.addEventListener('input', e => {
  setTemperature(parseInt(e.target.value));
});
modeSelect.addEventListener('change', e => {
  setMode(e.target.value);
});
autoSunChk.addEventListener('change', e => {
  if (e.target.checked) startAutoSun();
```

```
  else stopAutoSun();
});
geoBtn.addEventListener('click', getGeoSun);

enableControls(false);

function enableControls(on) {
  [powerBtn, brightnessSlider, temperatureSlider, modeSelect, autoSunChk, geoBtn].forEach(el => {
    el.disabled = !on;
  });
}

function updateConnectStatus(status) {
  const statusDiv = document.getElementById('connectStatus');
  const icon = document.getElementById('connectStatusIcon');
  const text = document.getElementById('connectStatusText');
  statusDiv.className = 'status ' + status;
  if (status === 'connected') {
    icon.style.background = '#41ea66';
    text.textContent = 'Підключено';
  } else if (status === 'connecting') {
    icon.style.background = '#ffa600';
    text.textContent = 'Підключення...';
  } else {
    icon.style.background = '#e44';
    text.textContent = 'Не підключено';
  }
  // для кнопки
  connectBtn.disabled = (status === 'connecting');
```

```
  connectBtn.classList.toggle('connecting', status === 'connecting');
}

function showStatus(msg, isErr=false) {
  const status = document.getElementById('status');
  status.textContent = msg;
  status.style.color = isErr ? '#ff6565' : '#a8f5af';
}

// ========== BLE SETTERS ============= //
async function setPower(on) {
  if (!isConnected) return;
  try {
    await charPower.writeValue(Uint8Array.of(on ? 1 : 0));
    powerBtn.classList.toggle('on', on);
    powerBtn.setAttribute('aria-pressed', on);
    showStatus(on ? 'Світло увімкнено' : 'Світло вимкнено');
  } catch (e) {
    showStatus('Помилка керування живленням: ' + e.message, true);
  }
}

async function setBrightness(val) {
  if (!isConnected) return;
  try {
    await charBrightness.writeValue(Uint8Array.of(val));
    showStatus('Яскравість: ' + val);
  } catch (e) {
    showStatus('BLE error: ' + e.message, true);
  }
```

```
}

async function setTemperature(val) {
  if (!isConnected) return;
  try {
    await charTemperature.writeValue(Uint8Array.of(val));
    showStatus('Температура: ' + val);
  } catch (e) {
    showStatus('BLE error: ' + e.message, true);
  }
}

async function setMode(mode) {
  if (!isConnected) return;
  try {
    const enc = new TextEncoder();
    await charMode.writeValue(enc.encode(mode));
    showStatus('Режим: ' + mode);
  } catch (e) {
    showStatus('BLE error: ' + e.message, true);
  }
}

// ============ SUNRISE/SUNSET AUTO ============ //
let autoSunTimerOn = null, autoSunTimerOff = null;
let autoSunActive = false;
let lastGeo = null;

function startAutoSun() {
  getGeoSun();
```

```
  autoSunActive = true;
  showStatus('Автоматизація активна');
}
function stopAutoSun() {
  autoSunActive = false;
  clearTimeout(autoSunTimerOn);
  clearTimeout(autoSunTimerOff);
  showStatus('Автоматизацію вимкнено');
}

async function getGeoSun() {
  if (!navigator.geolocation) {
    showStatus('Геолокація не підтримується', true);
    return;
  }
  showStatus('Визначаємо місцезнаходження...');
  navigator.geolocation.getCurrentPosition(async pos => {
    const { latitude, longitude } = pos.coords;
    lastGeo = { latitude, longitude };
    showStatus('Геолокацію визначено');
    await fetchSunriseSunset(latitude, longitude);
  }, err => {
    showStatus('Помилка геолокації: ' + err.message, true);
  });
}

async function fetchSunriseSunset(lat, lng) {
  showStatus('Отримуємо час сходу/заходу сонця...');
  try {
    const resp = await fetch(
```

```
      `https://api.sunrise-sunset.org/json?lat=${lat}&lng=${lng}&formatted=0`
    );
    const data = await resp.json();
    if (!data.results) throw new Error('no sun data');
    const sunrise = new Date(data.results.sunrise).getTime();
    const sunset  = new Date(data.results.sunset).getTime();
    scheduleSunTimers(sunrise, sunset);
    showStatus('Схід: ' + new Date(sunrise).toLocaleTimeString() +
               ', захід: ' + new Date(sunset).toLocaleTimeString());
  } catch (e) {
    showStatus('Не вдалося отримати час сонця: ' + e.message, true);
  }
}

function scheduleSunTimers(sunrise, sunset) {
  clearTimeout(autoSunTimerOn);
  clearTimeout(autoSunTimerOff);
  const now = Date.now();
  // Якщо сход уже був - плануємо на завтра
  let onMs  = (sunset  > now) ? (sunset  - now) : (sunset  - now + 24*3600*1000);
  let offMs = (sunrise > now) ? (sunrise - now) : (sunrise - now + 24*3600*1000);
  autoSunTimerOn  = setTimeout(() => { setPower(true);  if (autoSunActive)
updateSunTimers(); }, onMs);
  autoSunTimerOff = setTimeout(() => { setPower(false); if (autoSunActive)
updateSunTimers(); }, offMs);
}

function updateSunTimers() {
  // Через добу - оновити координати/час сонця
  if (lastGeo) fetchSunriseSunset(lastGeo.latitude, lastGeo.longitude);
```

```
}

// ================ THEME TOGGLE =============== //
const themeBtn = document.getElementById('themeBtn');
themeBtn.addEventListener('click', () => {
  document.body.classList.toggle('dark-theme');
  themeBtn.innerHTML = document.body.classList.contains('dark-theme') ? '🌙' : '🔆';
});

// Init
showStatus('Підключіть пристрій для керування.');
```

ДОДАТОК Г
(ОБОВ'ЯЗКОВИЙ)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА КОМУНІКАЦІЙНОГО ІНТЕРФЕЙСУ ДЛЯ МІКРОКОНТРОЛЕРА ARDUINO ЗА ТЕХНОЛОГІЄЮ BLUETOOTH

1. Таксономія BLE-додатків, їхні проблеми та рішення
2. Електрична схема підключення
3. Загальний вигляд користувацького інтерфейсу
4. Блок-схема алгоритму користувацького досвіду
5. Вебінтерфейс системи
6. Пошук пристроїв для з'єднання
7. Вибір пристрою для з'єднання
8. Елементи керування освітленням
9. Результати тестування статусної строки стану

Студент групи 2АКІТ-21б

Денис Щепленський

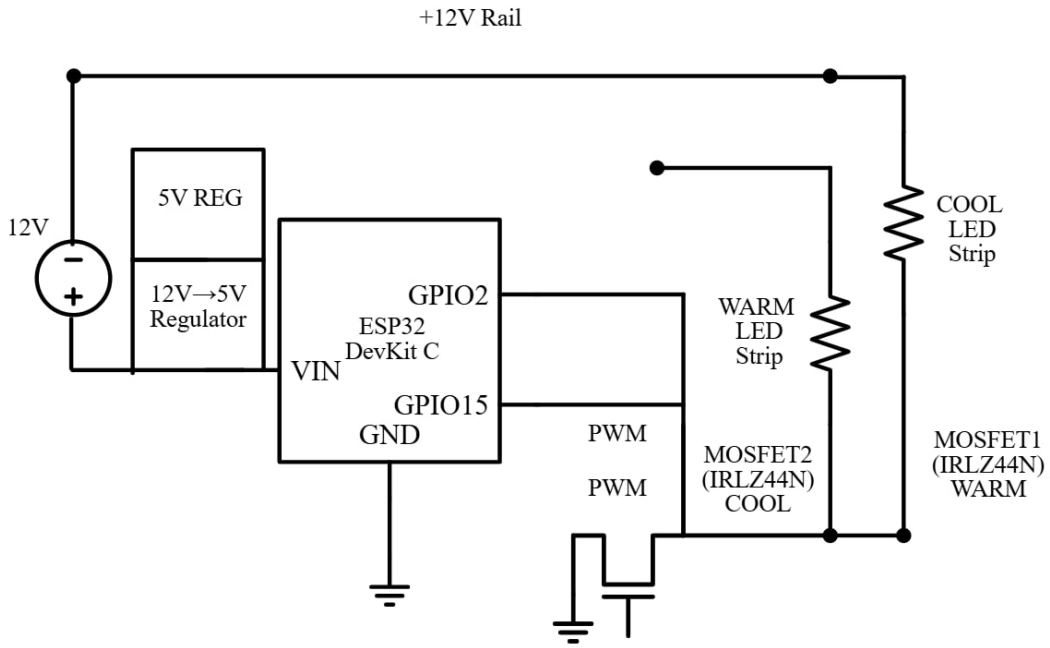
Керівник д.т.н., проф.

В'ячеслав КОВТУН

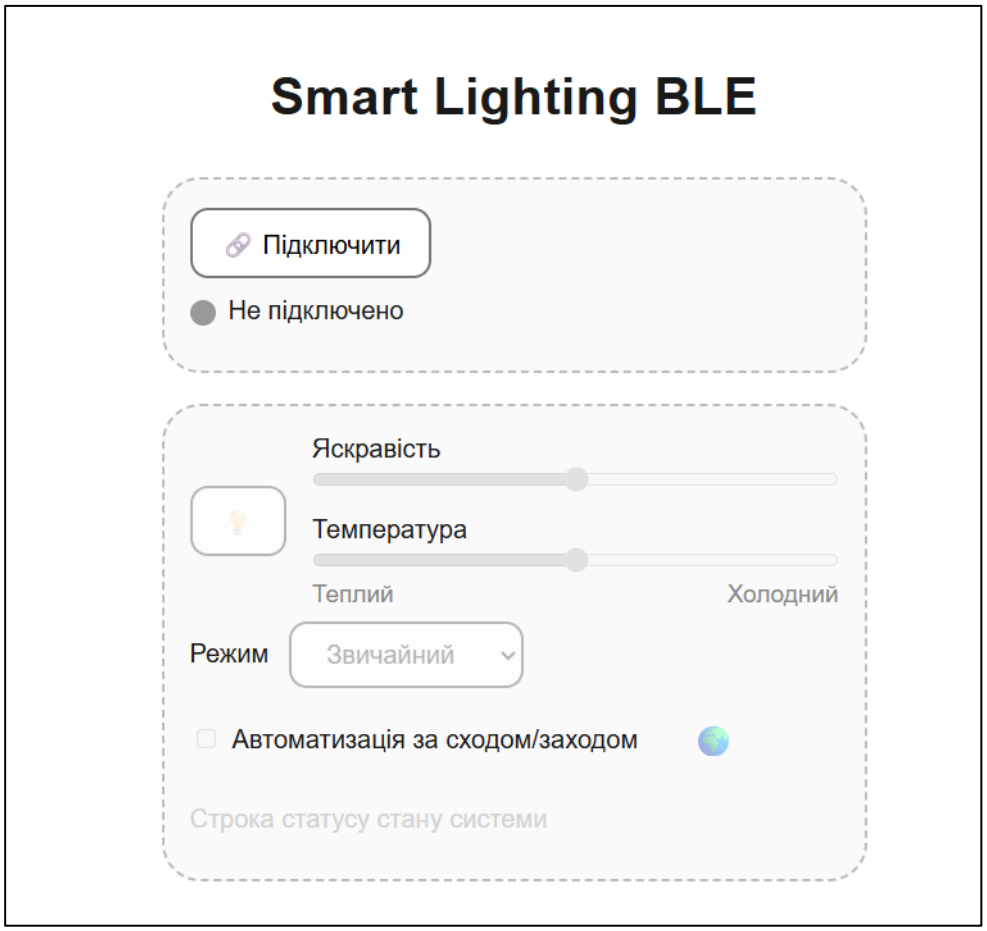
Вінниця 2025

ТАКСОНОМІЯ BLE-ДОДАТКІВ, ЇХНІ ПРОБЛЕМИ ТА РІШЕННЯ

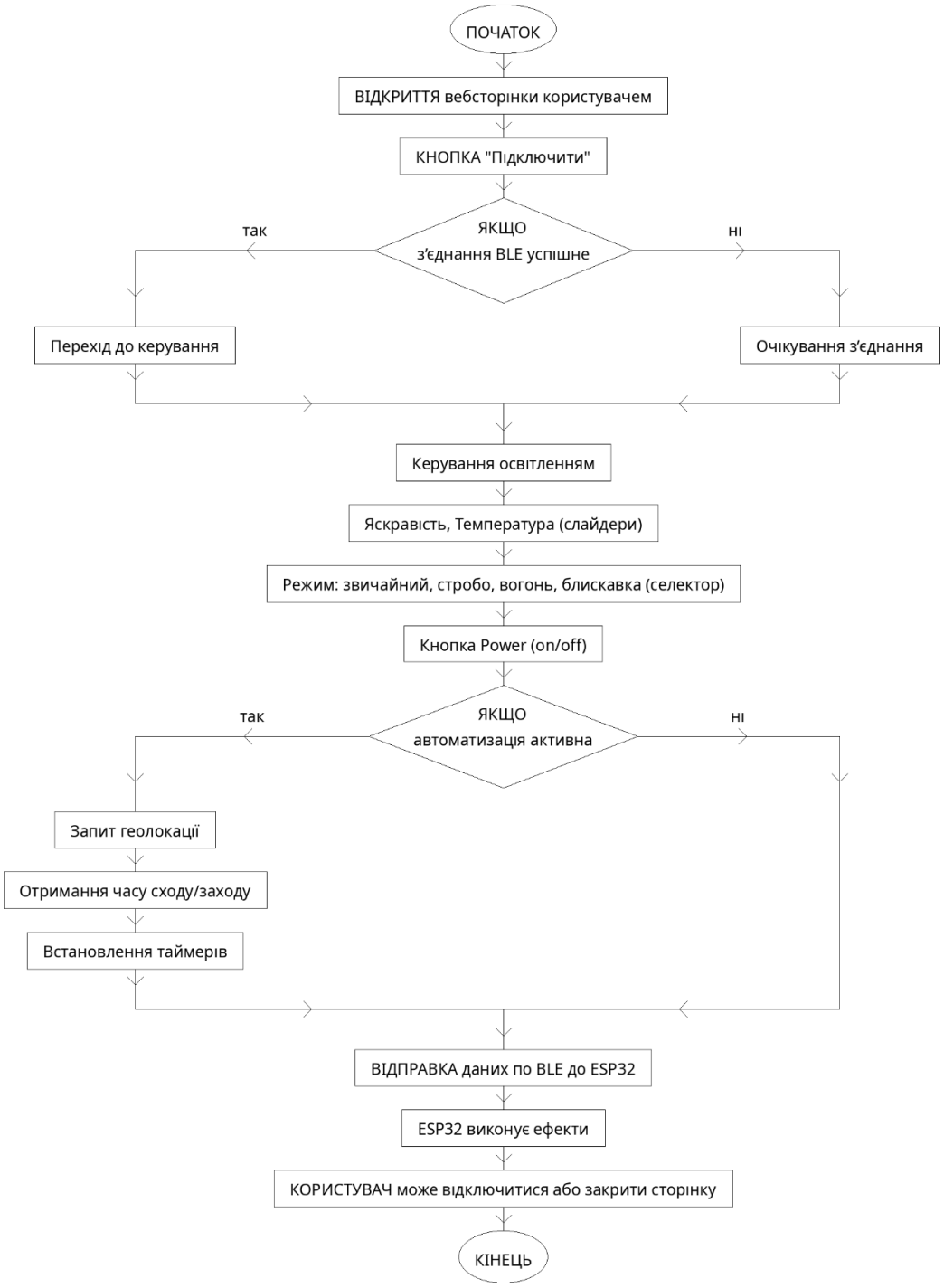
ЕЛЕКТРИЧНА СХЕМА ПІДКЛЮЧЕННЯ



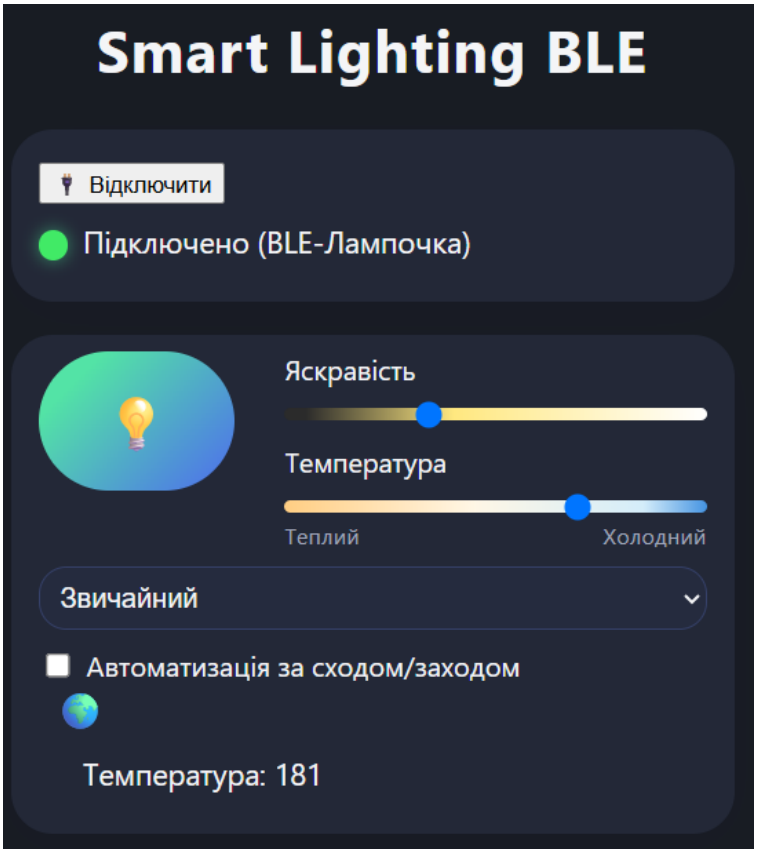
ЗАГАЛЬНИЙ ВИГЛЯД КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ



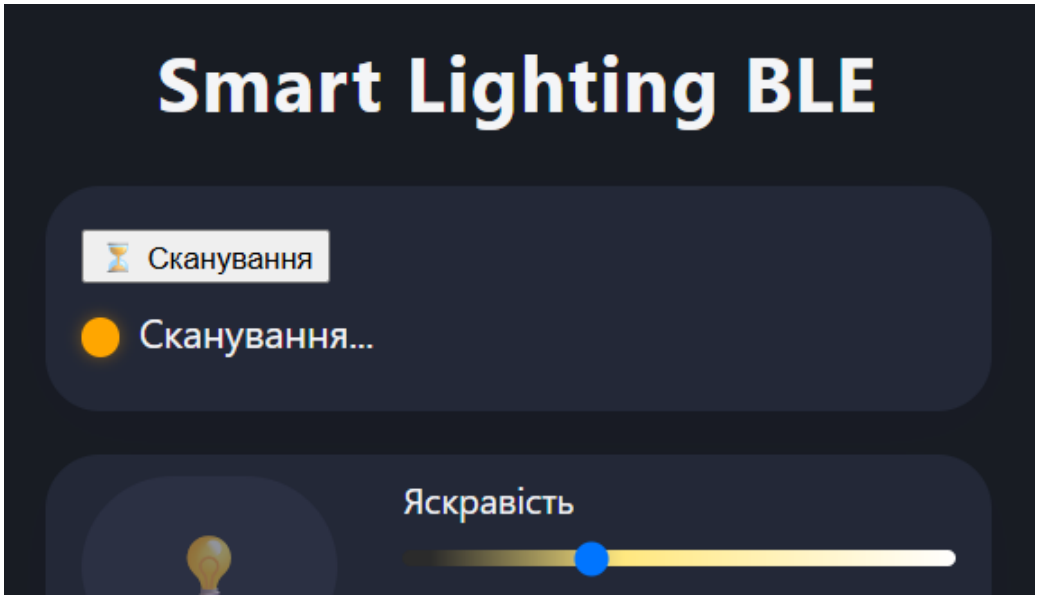
БЛОК-СХЕМА АЛГОРИТМУ КОРИСТУВАЦЬКОГО ДОСВІДУ



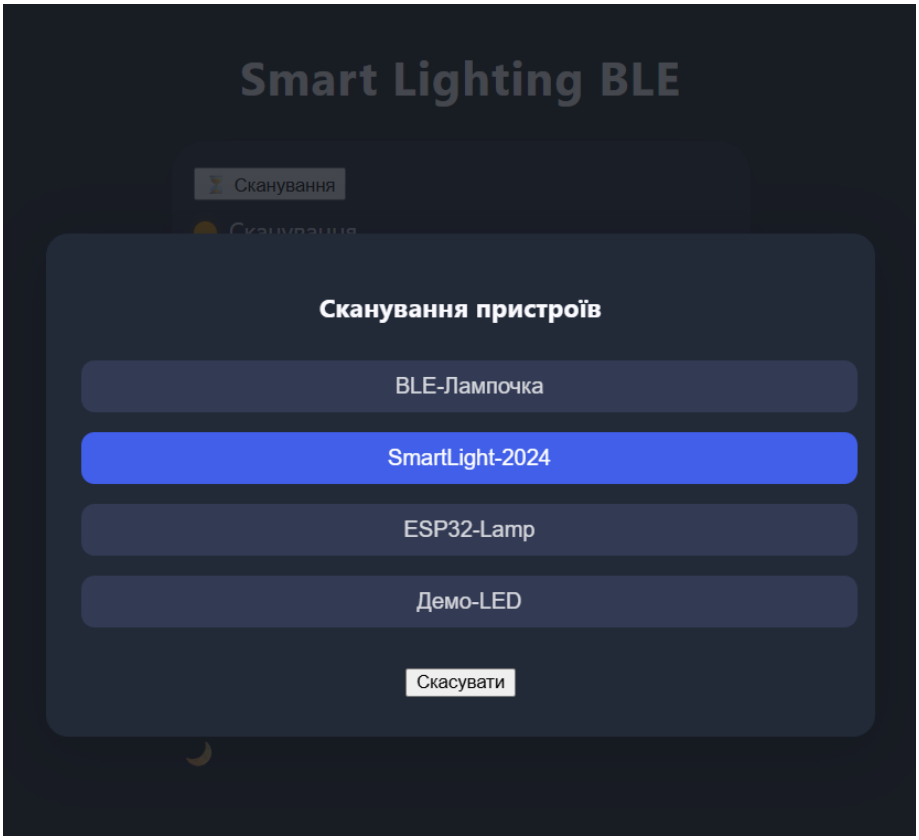
ВЕБІНТЕРФЕЙС СИСТЕМИ



ПОШУК ПРИСТРОЇВ ДЛЯ З'ЄДНАННЯ



ВИБІР ПРИСТРОЮ ДЛЯ З’ЄДНАННЯ



ЕЛЕМЕНТИ КЕРУВАННЯ ОСВІТЛЕННЯМ

РЕЗУЛЬТАТИ ТЕСТУВАННЯ СТАТУСНОЇ СТРОКИ СТАНУ СИСТЕМИ