МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри,
д-р техн. наук, проф.
_____ Ірина ЖУРАВСЬКА
« __ » _____ 202__ р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

АДАПТИВНЕ УПРАВЛІННЯ ОСВІТЛЕННЯМ В ПРИМІЩЕННІ ЗА ДОПОМОГОЮ ДАТЧИКІВ ТА АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ

Спеціальність 123 Комп'ютерна інженерія
Освітня програма «Комп'ютерна інженерія»

Здобувач _____Леонід КОЗЯВКО
підпис
«__» _____ 202__ р.

Керівник канд. фіз-мат. наук, доц. кафедри КІ_____Сергій ПУЗИРЬОВ
підпис
«__» _____ 202__ р.

Миколаїв – 2024

Завдання на виконання кваліфікаційної магістерської роботи

Факультет	Комп’ютерних наук
Кафедра	Комп’ютерної інженерії
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	123 Комп’ютерна інженерія
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри комп’ютерної інженерії
_____ Ірина ЖУРАВСЬКА
«___»_____ 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

_____Козявко Леоніда Олександровича_____
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи
_____
Адаптивне управління освітленням в приміщенні за допомогою датчиків та алгоритмів машинного навчання
_____
Затверджена наказом ректора ЧНУ ім. Петра Могили від «16» вересня 2024 р. № 236.

2. Строк представлення кваліфікаційної роботи «____» _____ 2024__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
_____
Очікуваним результатом роботи є: апаратне та програмне забезпечення адаптивної системи управлінням освітленням. Вхідними даними роботи є специфікація вимог, що описує характеристики зазначеного апаратного та програмного забезпечення._____

4. Перелік питань, що підлягають розробці _____
_____
1) дослідження основних систем освітлення;_____
2) аналіз систем адаптивного управління;_____
3) розробка апаратної частини пристрою;_____
4) розробка програмної частини пристрою;

5. Перелік графічних матеріалів

Презентація

6. Завдання до спеціальної частини
_____
_____
_____

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ _Сергій ПУЗИРЬОВ_____
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ _Леонід КОЗЯВКО____
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання «____» _____ 2024__ р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної магістерської роботи

Тема:___ Адаптивне управління освітленням в приміщенні за допомогою датчиків та алгоритмів машинного навчання.

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	01.09.2024	07.09.2024	Виконано
2.	Огляд літератури за темою роботи	08.09.2024	18.09.2024	Виконано
3.	Складання календарного плану КМР	19.09.2024	20.09.2024	Виконано
4.	Аналіз предметної області	21.09.2024	27.09.2024	Виконано
5.	Розробка проєктних рішень	28.09.2024	15.10.2024	Виконано
6.	Моделювання та конструювання АПЗ	16.10.2024	27.10.2024	Виконано
7.	Перевірка працездатності, тестування та апробація розробленого АПЗ, аналіз результатів тестування, розробка керівництва користувача	28.10.2024	20.11.2024	Виконано
8.	Відгук керівника КМР	21.11.2024	21.11.2024	Виконано
9.	Оформлення КМР та презентації	22.11.2024	27.11.2024	Виконано
10.	Попередній захист	28.11.2024	28.11.2024	Виконано
11.	Рецензування	29.11.2024	30.11.2024	Виконано
12.	Завершення оформлення КМР та презентації	01.12.2024	05.12.2024	Виконано
13.	Захист кваліфікаційної роботи	19.12.2024	19.12.2024	Виконано

Керівник роботи _____ __Сергій ПУЗИРЬОВ__
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ __Леонід КОЗЯВКО___
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

АНОТАЦІЯ

до кваліфікаційної магістерської роботи
«Адаптивне управління освітленням в приміщенні за допомогою датчиків та алгоритмів машинного навчання»
Здобувач гр. 605м Козявко Леонід Олександрович
Керівник: канд. фіз.-мат. наук, доцент кафедри КІ Сергій Пузирьов

В умовах сучасного світу питання ефективного використання енергоресурсів набуває особливого значення. Однією з ключових сфер, де можна значно оптимізувати енергоспоживання, є освітлення приміщень. Розробка адаптивної системи управління освітленням, яка автоматично регулює інтенсивність освітлення залежно від реальних умов у приміщенні та поведінки користувачів, сприятиме зниженню витрат на електроенергію та підвищенню комфорту.

Об'єкт дослідження – процес автоматизації управління освітленням у приміщеннях, що базується на інтеграції даних із датчиків.

Предмет дослідження – взаємодія між системою управління освітленням та її компонентами (датчики освітленості, руху, часу доби) та алгоритмами машинного навчання для оптимізації освітлення.

Мета роботи – розробити адаптивну систему управління освітленням на основі передових технологій датчиків та алгоритмів машинного навчання, яка забезпечить енергоефективність і комфорт, залежно від поточних умов та потреб користувачів.

Для досягнення мети поставлено такі завдання:

– провести аналіз сучасних систем адаптивного управління освітленням, виявити їх переваги та недоліки;

– розробити архітектуру системи, що включає алгоритми управління освітленням із використанням сенсорів та технологій комп'ютерного зору;

– реалізувати апаратну частину системи на основі мікроконтролера esp32-cam та відповідних датчиків;

– розробити програмну частину системи, включаючи мобільний додаток і серверну частину для обробки даних;

– провести тестування системи, проаналізувати результати та оцінити її ефективність.

Кваліфікаційна робота містить: перелік скорочень, вступ, чотири розділи, висновки, перелік джерел посилань та 2 додатків.

Вступ містить обґрунтування актуальності теми дослідження, об’єкт, предмет дослідження, мету та завдання, які необхідно виконати для досягнення поставленої мети.

У першому розділі виконано аналіз сучасних систем адаптивного управління освітленням, виявлено їх переваги й недоліки, а також визначено напрямки їхнього вдосконалення.

У другому розділі описано теоретичні основи роботи, включаючи алгоритми управління освітленням, технології датчиків, методи машинного навчання та інтеграцію їх в адаптивну систему.

У третьому розділі представлено процес розробки системи, включаючи вибір компонентів, проєктування апаратної частини на основі мікроконтролера ESP32-CAM, опис реалізації серверної частини, мобільного застосунку.

Четвертий розділ присвячено тестуванню розробленої системи, аналізу її роботи за різних умов, порівнянню отриманих результатів із існуючими рішеннями та оцінці її ефективності.

У висновках наведено основні результати виконаної кваліфікаційної роботи та зроблено висновки про досягнення поставленої мети.

Додатки містять код програмного забезпечення та інформацію про перевірку кваліфікаційної роботи.

Результати дослідження підтверджують ефективність запропонованої системи, що дозволяє знижувати енергоспоживання та підвищувати комфорт користувачів.

Кваліфікаційна робота містить 79 сторінок (без додатків), 38 рис., 7 табл., 22 літературних джерела, 2 додатки.

Ключові слова: світлодіод (LED), адаптивне освітлення, датчик освітленості, PIR-датчик, IoT (Інтернет речей), ESP32-CAM, енергоефективність, RGB-стрічка, автоматизація освітлення, користувацькі вподобання.

ASTRACT

of the Master’s Thesis

"Adaptive indoor lighting control using sensors and machine learning algorithms"

Applicant: Koziavko Leonid Oleksandrovich

Supervisor: PhD in Physical and Mathematical Sciences, Associate Professor of the Department of Computer Engineering

In the modern world, the issue of efficient energy use is of paramount importance. One of the key areas where energy consumption can be significantly optimized is indoor lighting. The development of an adaptive lighting control system that automatically adjusts lighting intensity based on real room conditions and user behavior will help reduce electricity costs and enhance comfort.

Object of study: The process of automating lighting control in indoor environments based on the integration of sensor data.

Subject of study: Interaction between the lighting control system and its components (light sensors, motion detectors, time-of-day sensors) and machine learning algorithms for lighting optimization.

Aim of the work: To develop an adaptive lighting control system based on advanced sensor technologies and machine learning algorithms, providing energy efficiency and comfort depending on current conditions and user needs.

To achieve the goal, the following tasks were set:

– analyze modern adaptive lighting control systems, identifying their advantages and disadvantages;

– develop the system architecture, including lighting control algorithms using sensors and computer vision technologies;

– implement the hardware part of the system based on the esp32-cam microcontroller and corresponding sensors;

– develop the software part of the system, including a mobile application and a server component for data processing;

– conduct system testing, analyze the results, and evaluate its effectiveness.

The qualification work includes: a list of abbreviations, an introduction, four chapters, conclusions, a list of references, and 2 appendices.

The introduction substantiates the relevance of the research topic, defines the object and subject of the study, and outlines the goals and tasks to be accomplished.

The first chapter provides an analysis of modern adaptive lighting control systems, highlights their advantages and disadvantages, and identifies directions for improvement.

The second chapter describes the theoretical foundations of the work, including lighting control algorithms, sensor technologies, machine learning methods, and their integration into an adaptive system.

The third chapter presents the system development process, including component selection, hardware design based on the ESP32-CAM microcontroller, and descriptions of the server and mobile application implementations.

The fourth chapter focuses on testing the developed system, analyzing its performance under various conditions, comparing the obtained results with existing solutions, and evaluating its effectiveness.

The conclusions summarize the main results of the qualification work and assess the achievement of the set goal.

The appendices include the software code and information on the qualification work's verification process.

The research results confirm the effectiveness of the proposed system, which allows for reduced energy consumption and enhanced user comfort.

The qualification work includes 79 pages (excluding appendices), 38 figures, 7 tables, 22 references, and 2 appendices.

Keywords: LED (Light Emitting Diode), adaptive lighting, light sensor, PIR sensor, IoT (Internet of Things), ESP32-CAM, energy efficiency, RGB strip, lighting automation, user preferences.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

IoT – Internet of Things

JSON – JavaScript Object Notation

LED – Light Emitting Diode

PIR – Passive Infrared Sensor

RGB – Red, Green, Blue

Wi-Fi – Wireless Fidelity

ВСТУП

В умовах сучасного світу питання ефективного використання енергоресурсів набуває особливого значення. Зростання вартості енергії, екологічні виклики та необхідність зниження викидів вуглекислого газу ставлять перед суспільством завдання пошуку нових рішень для оптимізації споживання енергії. Однією з ключових областей, де споживання електроенергії може бути значно зменшене, є освітлення в приміщеннях, яке займає значну частку у загальному енергоспоживанні як житлових, так і комерційних будівель.

Актуальність дослідження полягає в необхідності розробки адаптивної системи управління освітленням, яка дозволить автоматично регулювати інтенсивність освітлення залежно від реальних умов у приміщенні та поведінки користувачів. Така система сприятиме зниженню витрат на електроенергію та підвищенню комфорту, що є важливим як з практичної, так і з екологічної точки зору.

Наукове значення роботи полягає у використанні передових технологій датчиків і алгоритмів машинного навчання для розробки інтелектуальної системи освітлення. Це дозволить не тільки автоматизувати процес управління освітленням, але й створити нові підходи до оптимізації ресурсів у будівельній індустрії та інтеграції технологій «розумного будинку».

Стан проблеми на сьогодні демонструє наявність ряду комерційних рішень у сфері розумного освітлення. Однак більшість із них мають обмежені можливості адаптації до складних сценаріїв користування та не використовують повною мірою потенціал машинного навчання для автоматичної оптимізації освітлення.

Об'єкт дослідження – процес автоматизації управління освітленням в приміщеннях, які використовують автоматичне управління на основі даних з датчиків.

Предмет дослідження – взаємодія між системою управління освітленням та її компонентами (датчики освітленості, присутності, часу доби та алгоритми машинного навчання), що спрямована на оптимізацію енергоспоживання та підвищення комфорту користувачів.

Мета роботи – розробити системи адаптивного управління освітленням, яка використовує передові технології датчиків та машинного навчання для автоматизації й оптимізації та підвищення ефективності освітлення в приміщеннях залежно від поточних умов та потреб користувачів.

Основними *завданнями* роботи є:

– провести аналіз існуючих систем адаптивного управління освітленням та визначити їхні переваги й недоліки;

– розробити архітектуру системи управління освітленням, що використовує датчики освітленості та присутності;

– реалізувати алгоритми машинного навчання для автоматичної адаптації освітлення до умов приміщення;

– оцінити ефективність запропонованої системи шляхом тестування на реальних даних і порівняти її з існуючими рішеннями.

Робота пройшла апробацію під час XXI Міжнародної наукової конференції «Ольвійський форум» (Миколаїв, 20–23 червня 2024 р.).

Основні положення кваліфікаційної магістерської роботи опубліковані у збірнику матеріалів XXI Міжнародної наукової конференції «Ольвійський форум – 2024» [1].

1 АНАЛІТИЧНИЙ ОГЛЯД СИСТЕМ УПРАВЛІННЯ ОСВІТЛЕННЯМ

1.1 Огляд систем освітлення

Системи освітлення відіграють важливу роль у забезпеченні комфорту та продуктивності в приміщеннях, незалежно від їх призначення. Ефективне освітлення здатне створити приємну атмосферу, покращити умови роботи та зберегти здоров’я очей користувачів. Розвиток технологій у цій галузі дозволяє використовувати як традиційні системи освітлення, так і сучасні інноваційні рішення, які поступово змінюють підхід до управління освітленням.

Традиційні системи освітлення, які базуються на ручному керуванні за допомогою вимикачів, до сьогодні залишаються найбільш поширеними через свою простоту. Однак вони мають низку серйозних недоліків, таких як відсутність автоматизації, високе енергоспоживання та короткий термін служби. Це призводить до нераціонального використання енергоресурсів, значних фінансових витрат і, найголовніше, екологічного навантаження.

З іншого боку, системи освітлення з фіксованими правилами дозволяють часткову автоматизацію завдяки використанню датчиків руху та освітленості, але їхні можливості обмежені через використання жорстких алгоритмів. Наприклад, вони не враховують змін у поведінці користувачів чи умовах середовища. Таким чином, ці системи є проміжним етапом між традиційним підходом і повністю автоматизованими рішеннями.

Найбільш перспективним напрямком розвитку є розумні системи освітлення, які забезпечують гнучке управління, інтеграцію з іншими елементами «розумного будинку» та адаптацію до потреб користувачів. Розумні системи освітлення використовують датчики освітленості, руху, а також алгоритми машинного навчання, що дозволяє їм автоматично регулювати яскравість і колір світла

відповідно до зовнішніх умов і вподобань користувачів. Вони є енергоефективними та забезпечують високий рівень комфорту [2].

Таблиця – 1.1 Порівняння характеристик традиційних, фіксованих і розумних систем освітлення

Критерій	**Традиційні системи**	**Системи з фіксованими правилами**	**Розумні системи**
Автоматизація	Відсутня	Часткова	Повна
Енергоефективність	Низька	Середня	Висока
Врахування вподобань	Немає	Обмежене	Є
Екологічний вплив	Високий	Середній	Низький

Особливу увагу необхідно приділити екологічному впливу традиційних систем освітлення, оскільки це один із ключових аспектів, який стимулює перехід до адаптивних рішень.

Наприклад, на освітлення припадає до 20 % світового енергоспоживання [9], а використання неефективних джерел світла, таких як лампи розжарювання, створює значне навантаження на навколишнє середовище через високі викиди вуглекислого газу та обсяги відходів. Галогенні лампи та люмінесцентні світильники містять небезпечні речовини, як-от ртуть, які можуть забруднювати довкілля, якщо їх не утилізувати належним чином.

Заміна таких систем на енергоефективні рішення, зокрема LED-лампи або адаптивні системи, дозволяє суттєво знизити рівень викидів CO_2, а також забезпечити економічну доцільність за рахунок зменшення витрат на енергію.

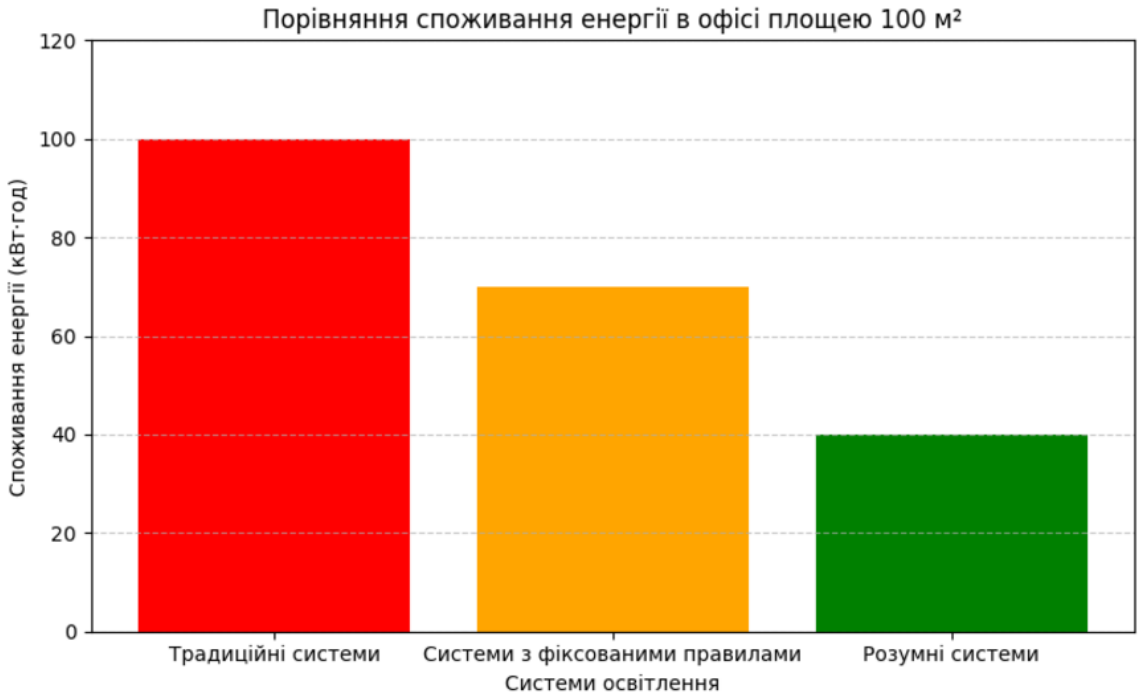


Рисунок 1.1 – Порівняння енергоспоживання різних систем освітлення в офісі площею 100 м²

Розумні системи освітлення є найбільш ефективними з точки зору екології, оскільки вони адаптуються до реальних умов приміщення, оптимізуючи рівень освітлення. Вони не лише скорочують енергоспоживання, але й мають тривалий термін служби, що зменшує кількість відходів. Наприклад, впровадження адаптивних систем у великому офісному центрі дозволяє знизити енергоспоживання на 30–50 %, що сприяє сталому розвитку. Таким чином, розвиток систем освітлення відображає не лише технологічний прогрес, але й відповідальність перед природою, стимулюючи впровадження інноваційних рішень для збереження ресурсів [3].

1.2 Огляд існуючих рішень у сфері систем адаптивного управління освітленням

У сучасному світі розумні системи управління освітленням стали незамінним елементом як для комерційних, так і для житлових приміщень. Їх головна мета – забезпечити оптимізацію енерговитрат та підвищити комфорт користувачів через автоматизацію освітлення. Сьогодні на ринку представлено низку комерційних рішень, таких як Philips Hue**,** Lutron**,** Caseta та інші.

1.2.1 Philips Hue

Philips Hue (рис. 1.2) є одним з найпопулярніших рішень у цій сфері. Це система, яка дозволяє користувачам контролювати освітлення через мобільний додаток або голосових асистентів (наприклад, Alexa або Google Assistant). Вона використовує датчики освітленості та руху для автоматичного ввімкнення та вимкнення світла [4].

Рисунок 1.2 – Philips Hue Starter Pack [20]

Користувачі можуть налаштовувати освітлення відповідно до своїх уподобань, створюючи індивідуальні сценарії для різних умов (робота, відпочинок тощо) [8]. Однак Philips Hue покладається на жорстко задані сценарії, які

потребують ручного налаштування. Це обмежує її здатність адаптуватися до змін у поведінці користувачів або зовнішніх умов.

1.2.2 Lutron

Lutron пропонує більш просунуті рішення для автоматизації освітлення, особливо у великих комерційних або офісних приміщеннях. *Lutron* дозволяє контролювати освітлення через датчики руху, освітленості та часу доби [5]. Це рішення більш комплексне і має більший потенціал для енергозбереження у великих приміщеннях *Lutron* дозволяє інтегрувати управління освітленням з іншими системами розумного будинку, такими як клімат-контроль або системи безпеки.

Рисунок 1.3 – Lutron Starter Kit [21]

Проте, як і *Philips Hue*, система *Lutron* також покладається на фіксовані правила і сценарії. У комерційних приміщеннях, де важлива гнучка адаптація освітлення до різних активностей або кількості присутніх, система може не досягати максимальних показників енергоефективності.

1.2.3 Caseta

Caseta – це ще одне популярне рішення, яке пропонує базову автоматизацію освітлення (рис. 1.4). Як і *Philips Hue* та *Lutron*, *Caseta* використовує датчики руху та освітленості для керування освітленням у приміщеннях. Система пропонує обмежену інтеграцію з іншими системами «розумного будинку» та не може автоматично адаптувати освітлення на основі змін у поведінці користувачів або змін зовнішніх умов [6].

Рисунок 1.4 – Caseta Lighting Kit [22]

Як і у випадку з іншими системами, керування освітленням у *Caseta* базується на фіксованих правилах, що не враховують динамічні зміни в умовах приміщення.

1.3 Порівняння фіксованих правил з адаптивними алгоритмами

Фіксовані правила управління освітленням є традиційним підходом, що широко використовується в сучасних комерційних системах. Вони передбачають автоматизацію на основі жорстко визначених умов, які задаються заздалегідь і залишаються незмінними під час експлуатації.

Наприклад, світло вмикається за сигналом датчика руху або вимикається через певний проміжок часу після виявлення відсутності користувача. Хоча такий підхід дозволяє вирішувати базові задачі автоматизації, він має суттєві обмеження.

Фіксовані правила ефективні лише для простих сценаріїв і не враховують динамічних змін у приміщенні.

Основні недоліки включають:

– **відсутність гнучкості:** система працює за жорстко заданими сценаріями, які не враховують динамічних змін, таких як поведінка користувачів або зовнішні умови. Наприклад, освітлення може залишатися на одному рівні, навіть якщо природне світло збільшується протягом дня;

– **недостатня персоналізація:** сценарії освітлення не враховують вподобання окремих користувачів. У випадку, коли в приміщенні перебувають декілька осіб, система не здатна налаштувати освітлення з урахуванням їхніх індивідуальних потреб;

– **низька енергоефективність:** неврахування змінних факторів призводить до нераціонального використання енергії. Наприклад, світло може залишатися ввімкненим у порожніх приміщеннях або не вимикатися при достатньому рівні природного освітлення.

На відміну від фіксованих правил, адаптивні алгоритми використовують дані з датчиків і алгоритми машинного навчання для динамічного налаштування параметрів освітлення. Це дозволяє системі автоматично реагувати на змінні умови та підлаштовуватися під потреби користувачів у реальному часі.

Основні переваги адаптивних алгоритмів:

– **гнучкість:** система здатна змінювати рівень яскравості або колірну температуру залежно від змін у рівні природного освітлення, часу доби чи активності користувачів;

– **персоналізація:** використання алгоритмів машинного навчання дозволяє враховувати індивідуальні вподобання користувачів. Наприклад, система може запам'ятовувати налаштування освітлення кожного користувача та автоматично застосовувати їх під час наступного використання;

– **підвищена енергоефективність:** Завдяки постійній адаптації до умов, система використовує мінімальну кількість енергії для підтримання комфортного рівня освітлення.

Фіксовані правила є базовим етапом автоматизації освітлення, але їхня статичність обмежує можливості оптимізації та персоналізації. Натомість адаптивні алгоритми демонструють значний потенціал завдяки своїй гнучкості, точності та енергоефективності. Перехід від фіксованих правил до адаптивних систем є важливим кроком у розвитку розумного освітлення.

1.4 Порівняльний аналіз існуючих рішень

Для оцінки ефективності комерційних систем управління освітленням було проведено порівняння за такими ключовими критеріями:

1) типи використовуваних датчиків;
2) алгоритми управління;
3) енергоефективність;
4) зручність для користувачів.

Таблиця 1.2 – Порівняльний аналіз систем

Критерій	**Philips Hue**	**Lutron**	**Caseta**	**Запропонована система**
Типи датчиків	Освітленості, руху	Освітленості, руху, часу доби	Руху, освітленості	Освітленості, руху, часу доби, камера (включається за сигналом з датчика руху)
Алгоритми управління	Фіксовані правила	Фіксовані правила	Фіксовані правила	Алгоритми машинного навчання, комп'ютерний зір
Енергоефективність	Середня	Висока	Середня	Дуже висока (адаптивне управління, оптимізація на основі даних з камери)
Зручність для користувача	Висока (через мобільні додатки та голосові команди)	Середня (обмежена гнучкість налаштувань)	Середня (основні функції автоматизації)	Висока (автоматична адаптація, мобільний додаток, персоналізація через розпізнавання особи)

Проведений аналіз показує, що комерційні системи (Philips Hue, Lutron, Caseta) мають певні переваги, але їхні фіксовані алгоритми не дозволяють досягти максимальної енергоефективності та персоналізації.

Запропонована система усуває ці недоліки завдяки використанню адаптивних алгоритмів, що дозволяє:

– оптимізувати споживання енергії;

– автоматично підлаштовувати освітлення до змінних умов;

– враховувати індивідуальні вподобання користувачів.

Це робить запропоновану систему конкурентоздатною у порівнянні з існуючими рішеннями, забезпечуючи новий рівень комфорту та ефективності.

1.5 Виявлення недоліків та формулювання вимог до нової системи

На основі порівняльного аналізу виявлено низку недоліків існуючих комерційних систем управління освітленням:

– відсутність можливості адаптивного управління, що базується на поведінці користувачів та умовах середовища;

– обмежена енергоефективність через використання фіксованих алгоритмів;

– відсутність персоналізації освітлення, враховуючи вподобання конкретних користувачів.

Для подолання цих недоліків нова система повинна відповідати наступним вимогам:

1) апаратні вимоги:

– використання датчиків освітленості, присутності та часу доби для точного моніторингу умов в приміщенні;

– використання камери для ідентифікації користувачів після спрацювання датчика руху. Камера активується лише у разі фіксації руху і дозволяє розпізнати людину, яка зайшла в приміщення;

– використання мікроконтролера для збору даних і управління освітленням;

2) програмні вимоги:

– впровадження алгоритмів машинного навчання для автоматичного налаштування освітлення [11];

– використання алгоритмів комп'ютерного зору для розпізнавання осіб та ідентифікації користувачів, з метою вибору налаштувань освітлення згідно з їхніми вподобаннями;

3) функціональні вимоги:

– адаптивне управління освітленням на основі змін у приміщенні, включаючи вподобання користувачів;

– дистанційне управління через мобільний додаток;

– персоналізація освітлення для кожного користувача, залежно від його вподобань, завантажених у мобільний застосунок;

4) енергоефективність та економічні вимоги:

– оптимізація енергоспоживання для мінімізації витрат на освітлення;

– зниження вуглецевого сліду шляхом зменшення споживання енергії.

Нова система забезпечить підвищену гнучкість, енергоефективність і комфорт завдяки персоналізації освітлення під конкретні потреби користувачів.

Висновки до розділу 1

У першому розділі виконано всебічний огляд сучасних систем управління освітленням, починаючи від традиційних підходів, які використовують ручне управління, і закінчуючи сучасними розумними системами. Аналіз продемонстрував, що більшість комерційних рішень мають суттєві недоліки, зокрема низький рівень адаптивності, обмежені можливості інтеграції з іншими системами та недостатню персоналізацію.

Існуючі системи, як-от Philips Hue, Lutron і Caseta, переважно покладаються на фіксовані алгоритми, що не дозволяє їм враховувати змінні умови середовища чи індивідуальні потреби користувачів. Це призводить до зайвого енергоспоживання та зниження рівня комфорту.

Для вирішення цих проблем було запропоновано адаптивну систему управління освітленням, яка базується на використанні алгоритмів машинного навчання, датчиків руху та освітленості, а також технологій комп’ютерного зору. Така система дозволяє динамічно налаштовувати параметри освітлення відповідно до поточних умов і потреб користувачів. Інтеграція з мобільним додатком забезпечує високий рівень персоналізації, що дає змогу зберігати індивідуальні налаштування та автоматично застосовувати їх.

Крім того, запропонована система підвищує енергоефективність, зменшуючи витрати електроенергії та одночасно знижуючи екологічний вплив.

Таким чином, запропоноване рішення не лише враховує сучасні вимоги до енергоефективності та комфорту, але й відповідає тенденціям інтеграції інтелектуальних технологій у повсякденне життя. Ця система має потенціал стати інноваційним рішенням, яке забезпечує якісно новий рівень автоматизації освітлення.

2 МАТЕМАТИЧНІ МЕТОДИ ТА ПРОЄКТУВАННЯ СИСТЕМИ, ЩО РОЗРОБЛЯЄТЬСЯ

2.1 Вибір математичних методів для реалізації системи

Адаптивне управління освітленням у приміщеннях потребує ефективних математичних методів, які здатні аналізувати вхідні дані та приймати рішення у реальному часі. Основним алгоритмом, який було обрано для реалізації цієї системи, є алгоритм дерево рішень (Decision Trees). Цей алгоритм є одним із найпопулярніших методів машинного навчання, який дозволяє будувати модель класифікації на основі логічних умов, що відповідають структурі дерева.

2.1.1 Теоретичні основи обраних математичних методів

Алгоритм дерево рішень (Decision Trees) належить до класу керованого навчання. Він є ефективним для розв’язання задач класифікації та регресії, оскільки дозволяє приймати рішення, розділяючи дані на підмножини на основі певних умов. В алгоритмі кожен вузол дерева відповідає за розділення на основі певного критерію, а листові вузли вказують на остаточні рішення.

Модель дерева будується шляхом ітеративного поділу даних на підмножини, де кожне розділення базується на максимальній інформаційній приростності або мінімізації Gini-індексу. Це дозволяє ефективно класифікувати дані на основі вхідних змінних, які впливають на прийняття рішень.

Основні переваги даного алгоритму для адаптивного управління освітленням:

– простота інтерпретації. Дерево рішень легко візуалізується та інтерпретується, що дозволяє зрозуміти, які змінні та критерії впливають на прийняття рішень у системі;

– швидке прийняття рішень. Алгоритм здатен швидко реагувати на змінні умови, що є критичним для адаптивного управління освітленням у реальному часі;

– гнучкість. Дерево рішень може бути використані для обробки як категоричних, так і числових даних, що дозволяє ефективно інтегрувати дані з різних сенсорів (датчики руху, освітленості, камери).

2.1.2 Основні змінні, що впливають на роботу алгоритму

У процесі адаптивного управління освітленням використовуються наступні змінні:

1) **присутність користувача:** якщо користувач присутній, система активується та налаштовує освітлення.

Ця змінна є першою в умовах дерева рішень, оскільки наявність або відсутність користувача є основним фактором для прийняття рішення щодо увімкнення або вимкнення світла.

2) **рівень природного освітлення:** вимірюється рівень природного світла у приміщенні. Якщо рівень освітленості достатній, алгоритм вирішує не вмикати додаткове освітлення.

Ця змінна є критичною для забезпечення енергоефективності, оскільки дозволяє зменшити споживання електроенергії, коли природного світла достатньо.

3) **вподобання користувачів:** кожен користувач може мати свої вподобання щодо рівня яскравості та кольорової температури освітлення. Алгоритм враховує ці вподобання, щоб забезпечити комфортні умови освітлення для конкретного користувача.

Якщо в приміщенні присутні кілька користувачів, алгоритм може використовувати пріоритети або компромісне рішення для налаштування освітлення.

4) **час доби:** час доби є важливою змінною, оскільки він впливає на налаштування кольорової температури світла (тепле світло ввечері, холодне –

вдень). Алгоритм використовує цю змінну для автоматичного налаштування освітлення відповідно до біологічних ритмів користувача.

2.1.3 Здатність алгоритму ефективно інтегрувати дані з різних сенсорів

Дерево рішень дозволяє легко інтегрувати дані з різних сенсорів та камер, оскільки кожна змінна може бути окремою умовою в дереві.

Наприклад:

Датчики руху вказують на присутність користувача, що є початковою умовою для роботи алгоритму.

Датчики освітленості вимірюють рівень природного світла, і це значення використовується для прийняття рішень щодо налаштування яскравості.

Камери дозволяють ідентифікувати конкретного користувача, що допомагає налаштувати освітлення на основі його особистих вподобань.

Алгоритм здатен адаптуватися до динамічних змін у приміщенні, аналізуючи дані у режимі реального часу та змінюючи параметри освітлення відповідно до нових умов. Завдяки цьому, система забезпечує ефективне використання енергії та комфортні умови для користувачів.

Алгоритм здатен інтегрувати дані з різних сенсорів та забезпечувати оптимальні умови освітлення у приміщенні, що сприяє економії енергії та підвищенню комфорту користувачів.

2.2 Розробка блок-схем алгоритму

Покроковий підхід до створення алгоритму забезпечує структуроване та ефективне вирішення завдань, пов'язаних з адаптацією освітлення відповідно до змінних умов у приміщенні.

2.2.1 Загальна блок-схема алгоритму

Побудова загальної блок-схеми є першим етапом розробки алгоритму. Основна мета цього етапу – визначити ключові етапи процесу управління освітленням та зрозуміти, що саме потрібно зробити, щоб система могла адаптувати освітлення у приміщенні.

Основні етапи загальної блок-схеми:

1) виявлення присутності користувача;
2) ідентифікація користувача;
3) аналіз рівня зовнішнього освітлення;
4) адаптація освітлення на основі вподобань користувача.

Рисунок 2.1 – Загальна блок-схема алгоритму адаптивного управління освітленням

2.2.2 Деталізація кожного блоку алгоритму

Другий етап розробки алгоритму передбачає детальну розробку кожного блоку загальної схеми. Цей етап визначає конкретні дії, які необхідно виконати для реалізації кожного блоку.

Блок «Виявлення присутності користувача»:

Процес виявлення починається з зчитування даних з датчика руху, підключеного до мікроконтролера.

Якщо датчик виявляє рух, процесор передає сигнал про присутність користувача у кімнаті.

Рисунок 2.2 – Детальна блок-схема алгоритму (Блок «Виявлення присутності користувача») частина 1

Блок «Ідентифікація користувача»:

Після підтвердження присутності камера вмикається для ідентифікації користувача.

Використовується алгоритм комп’ютерного зору для обробки зображення і перевірки даних щодо користувача.

Якщо користувача ідентифіковано, система зберігає дані його вподобань і переходить до наступного етапу. Якщо ідентифікація невдала, обирається режим за замовчуванням або налаштування користувача з першим пріоритетом.

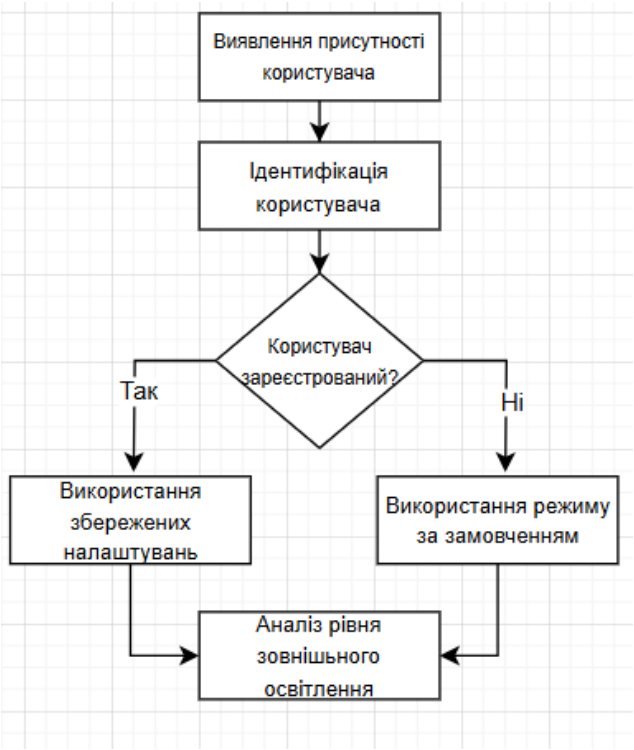


Рисунок 2.3 – Детальна блок-схема алгоритму (Блок «Ідентифікація користувача») частина 2

Блок «Аналіз рівня зовнішнього освітлення»:

Датчик вимірює рівень природного освітлення, і отримані дані передаються на мікроконтролер для обробки.

Після чого перевіряється, чи рівень природного світла перевищує певний поріг. Якщо так, додаткове освітлення не вмикається.

Якщо рівень природного світла менший за поріг, алгоритм вирішує, яке додаткове освітлення необхідне, щоб забезпечити комфортні умови у приміщенні.

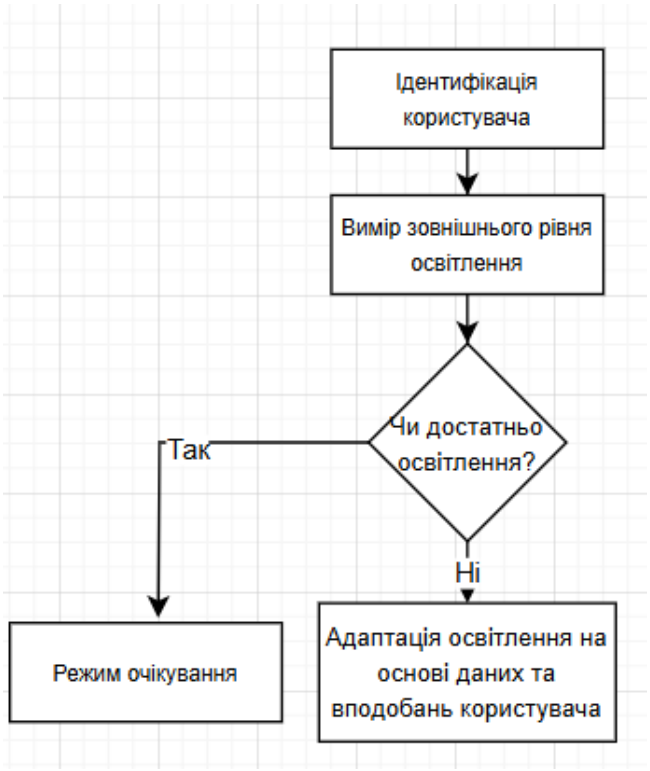


Рисунок 2.4 – Детальна блок-схема алгоритму (Блок «Аналіз рівня зовнішнього освітлення») частина 3

Блок «Налаштування освітлення на основі вподобань користувача»:

Система налаштовує рівень яскравості та кольорову температуру відповідно до уподобань користувача або режиму за замовчуванням.

Якщо в кімнаті є кілька користувачів, може використовується компромісне налаштування, яке враховує уподобання кожного користувача. Крім того, можливе використання пріоритетного режиму, при якому застосовуються налаштування користувача, який має найбільший пріоритет.

Рисунок 2.5 – Детальна блок-схема алгоритму (Блок «Налаштування освітлення») частина 4

Побудова блок-схем алгоритму адаптивного управління освітленням забезпечує структурований підхід до прийняття рішень у системі. Покрокова деталізація кожного блоку дозволяє ефективно інтегрувати дані з сенсорів та камер, а також враховувати індивідуальні вподобання користувачів. Це підвищує точність і гнучкість системи, забезпечуючи комфортне освітлення та зниження енерговитрат.

2.3 Математичне моделювання системи

Математичне моделювання є основним етапом розробки системи адаптивного управління освітленням, оскільки воно дозволяє формалізувати

процеси прийняття рішень на основі вхідних даних, таких як рівень природного освітлення, присутність користувача та його вподобання.

Моделювання системи базується на побудові математичної моделі, яка використовує змінні вхідних даних для автоматизації процесу прийняття рішень. Ця модель дозволяє інтегрувати дані з різних сенсорів та враховувати вподобання користувачів для досягнення оптимальних умов освітлення у приміщенні.

2.3.1 Побудова математичної моделі системи

Модель адаптивного управління освітленням складається з кількох основних компонентів:

Змінні вхідних даних:

– рівень освітленості (**L**): вимірюється датчиками освітленості (LDR) та представляє рівень природного світла у приміщенні;

– пріоритет користувача (**P**): визначає важливість уподобань користувача щодо яскравості та кольорової температури;

– час доби (**T**): враховується для автоматичної зміни кольорової температури світла (наприклад, тепле світло у вечірній час, холодне – вдень);

– присутність користувача (**U**): визначає, чи є у кімнаті користувач, і чи потрібно вмикати освітлення.

Далі розглянемо основні математичні функції для оцінки умов прийняття рішень.

1) Функція визначення середнього рівня освітлення

Функція обчислює середній рівень освітленості у приміщенні, щоб визначити, чи потрібно вмикати додаткове освітлення. Це допомагає зрозуміти загальний рівень світла, який надходить у кімнату з вулиці або від інших джерел.

Формула виглядає так:

$$L_{avg} = \frac{\sum_{n-1}^{n} L_i}{n}, \quad (2.1)$$

де L_{avg} – середній рівень освітленості;

L_i – виміряне значення освітленості з *i*-го датчика або за *i*-й момент часу;

n – загальна кількість вимірів або сенсорів.

Функція працює таким чином:

Система зчитує рівні освітленості з одного або кількох датчиків

Потім усі значення освітленості підсумовуються та діляться на кількість вимірів, щоб отримати середній рівень.

Отримане середнє значення дозволяє системі зрозуміти, чи достатньо природного світла в кімнаті. Якщо середній рівень перевищує певний поріг, додаткове освітлення не вмикається.

2) Функція компромісу між вподобаннями користувачів

Ця функція використовується, коли у кімнаті одночасно перебуває кілька користувачів, і кожен із них має різні вподобання щодо рівня освітлення. Функція обчислює середнє значення між вподобаннями користувачів, щоб досягти компромісу в налаштуванні світла.

Формула має наступний вигляд:

$$P_{avg} = \frac{\sum_{j-1}^{m} P_j}{m}, \quad (2.2)$$

де P_{avg} – середнє значення вподобань яскравості;

P_j – уподобання j-го користувача щодо рівня яскравості;

m – загальна кількість користувачів у кімнаті.

Функція працює таким чином:

Система зчитує уподобання кожного користувача, які можуть включати бажаний рівень яскравості та колірної температури.

Всі значення вподобань підсумовуються і діляться на кількість користувачів, щоб знайти середнє значення, яке буде використане як компромісне налаштування освітлення.

Функція компромісу дозволяє забезпечити оптимальний рівень освітлення, який враховує індивідуальні вподобання користувачів, але не враховує пріоритети. Це дозволяє зберегти комфортні умови для всіх користувачів одночасно, забезпечуючи гнучку адаптацію освітлення в приміщенні.

3) Функція визначення оптимального рівня освітлення

Ця функція поєднує середній рівень освітленості та середні вподобання користувачів для визначення оптимального рівня штучного освітлення, який буде забезпечувати комфорт у кімнаті.

Формула виглядає так:

$$L_{opt} = L_{avg} + \alpha(P_{avg} - L_{avg}), \quad (2.3)$$

де L_{opt} – оптимальний рівень освітлення в приміщенні;

L_{avg} – середній рівень поточного природного освітлення;

P_{avg} – середнє значення вподобань користувачів;

α – коефіцієнт адаптації, який враховує, наскільки сильно повинні враховуватися вподобання користувачів. Цей коефіцієнт може варіюватися в залежності від системи та бажаної гнучкості адаптації (наприклад, значення від 0 до 1).

Функція працює таким чином:

Функція починає з середнього рівня освітленості у кімнаті (2.1).

Потім враховуються середні вподобання користувачів (2.2), а коефіцієнт α α визначає, наскільки система повинна адаптуватися до цих вподобань. Якщо α = 0, система повністю ігнорує вподобання користувачів і залишає рівень освітлення таким, як є. Якщо α=1, система повністю враховує середні вподобання користувачів.

Формула обчислює оптимальний рівень освітлення, який є компромісом між поточним природним освітленням та середнім рівнем вподобань користувачів, враховуючи коефіцієнт адаптації.

2.3.2 Загальний підхід до застосування математичних функцій

Ці функції використовуються у поєднанні для забезпечення адаптивного управління освітленням:

Функція середнього рівня освітлення (2.1) дозволяє зрозуміти поточні умови.

Функція компромісу між вподобаннями (2.2) забезпечує гнучкість системи при наявності кількох користувачів.

Функція оптимального рівня освітлення (2.3) поєднує обидві попередні функції, щоб знайти найкраще рішення для налаштування штучного освітлення.

Це підвищує ефективність та комфортність системи, знижуючи енерговитрати та забезпечуючи індивідуальний підхід до кожного користувача.

Висновок до розділу 2

У цьому розділі було розглянуто математичні моделі та алгоритми, що використовуються для реалізації адаптивного управління освітленням у приміщенні. Зокрема, описано покрокову побудову алгоритмів, які враховують виявлення користувачів, аналіз поточного рівня освітленості та адаптацію освітлення на основі індивідуальних вподобань користувачів.

Основною технологією стала модель рішення дерев, яка дозволяє ефективно класифікувати вхідні дані та приймати оптимальні рішення для налаштування освітлення. Уведення коефіцієнта адаптації та компромісного підходу до налаштування освітлення дозволяє забезпечити комфорт для користувачів, враховуючи як їхні уподобання, так і поточні умови освітлення.

Система демонструє високу гнучкість та енергоефективність, оскільки активує додаткове освітлення лише за потреби, а адаптація налаштувань базується на реальних даних. Розроблені блок-схеми алгоритмів ілюструють логіку системи, яка легко адаптується до змінних умов та потреб користувачів, забезпечуючи персоналізацію та комфорт.

3 РЕАЛІЗАЦІЯ АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (АПЗ)

3.1 Апаратне забезпечення пристрою

Апаратна частина системи автоматичного управління освітленням базується на поєднанні надійних і ефективних компонентів, які забезпечують стабільну роботу, точність даних і простоту інтеграції. Нижче наведено детальний опис обраних компонентів, їх характеристик, ролі у системі та обґрунтування вибору.

3.1.1 Мікроконтролер

ESP32-CAM OV2640 (рис. 3.1) – це потужний мікроконтролер, який об'єднує функції збору даних, обробки зображень і бездротової передачі інформації. Завдяки інтегрованому модулю камери OV2640, пристрій здатний виконувати завдання з розпізнавання обличь (face detection) та передачі фото або відео для подальшої обробки. ESP32-CAM ідеально підходить для проєктів, що потребують компактності, багатофункціональності та високої продуктивності.

Основним завданням ESP32-CAM у нашій системі є збір інформації від датчиків руху та освітленості, активація камери після виявлення присутності, виконання первинного розпізнавання обличь та передача отриманих зображень або відео на серверну частину для ідентифікації користувача. Цей підхід дозволяє розподілити обчислювальне навантаження, зменшуючи ресурси, необхідні на пристрої, та забезпечує більш гнучку обробку даних.

Таблиця 3.1 – Характеристики ESP32-CAM OV2640

Параметр	**Значення**
Процесор	Dual-core Tensilica LX6
Камера	OV2640 (2 Мп)
Роздільна здатність зображень	До 1600 × 1200 пікселів

Параметр	**Значення**
Пам’ять PSRAM	4 МБ
Флеш-пам’ять	8 МБ
Інтерфейси	Wi-Fi 802.11 b/g/n, Bluetooth 4.2
Живлення	5 В
Габарити	27 мм × 40.5 мм

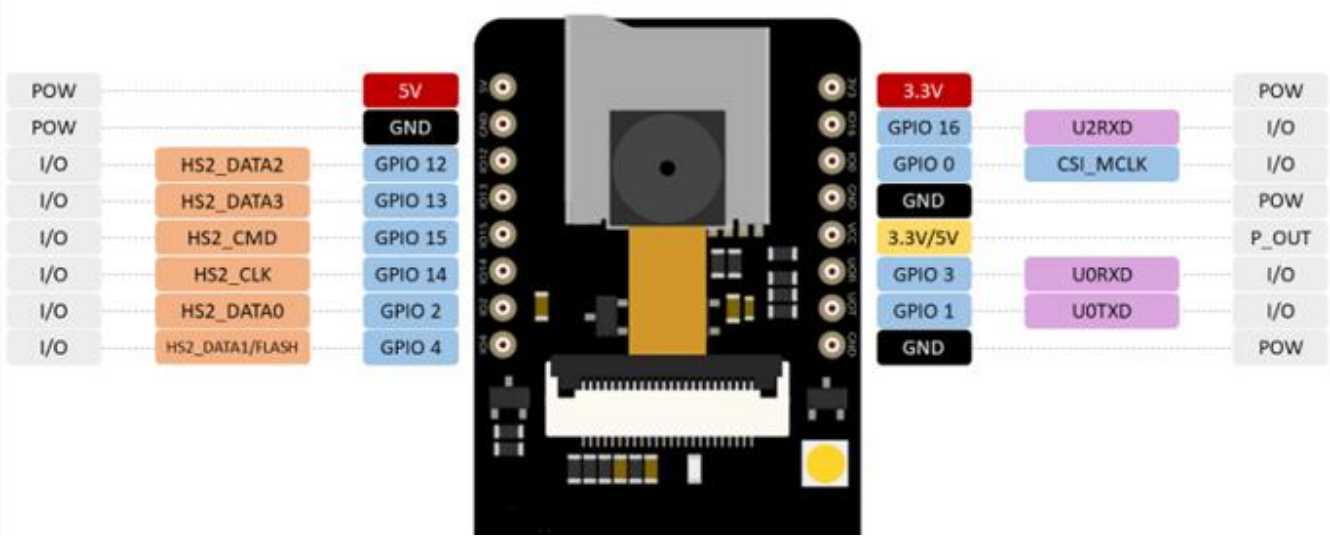


Рисунок 3.1 – Схема ESP32-CAM [12]

3.1.2 PIR-датчик HC-SR501

PIR-датчик HC-SR501 (Passive InfraRed Sensor) (рис. 3.2) є одним із ключових елементів системи, який забезпечує виявлення руху в приміщенні. Він працює за принципом пасивного інфрачервоного виявлення, реагуючи на зміни теплового випромінювання, що виникають під час руху людини. PIR-датчик HC-SR501 відзначається точністю, низьким енергоспоживанням та зручністю вбудови в автоматизовані рішення.

Таблиця 3.2 – Характеристики HC-SR501 [13]

Параметр	**Значення**
Діапазон виявлення	До 7 м
Кут виявлення	120°
Робоча напруга	4.5–20 В
Струм у спокої	<50 мкА
Регулювання часу затримки	0.3–200 секунд
Розміри	32 мм × 24 мм

Таблиця 3.3 – Конфігурація виводів модуля PIR-датчика [13]

Номер виводу	**Назва виводу**	**Опис**
1	Vcc	Вхідна напруга +5 В для типових застосувань. Може варіюватися від 4.5 В до 12 В.
2	Вихід High/Low (Dout)	Цифровий імпульс високого рівня (3.3 В) при спрацьовуванні (рух виявлено), низького рівня (0 В) у стані спокою (руху немає).
3	Ground	Підключення до заземлення схеми.

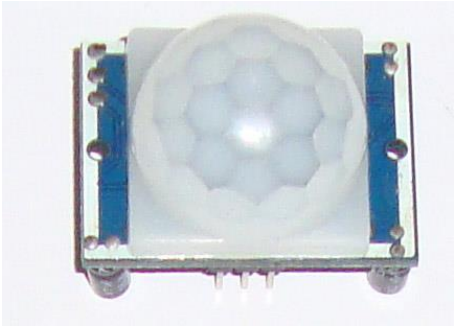


Рисунок 3.2 – HC-SR501 [13]

\

HC-SR501 працює як тригер для активації функцій системи. Після виявлення руху датчик передає сигнал на ESP32, який вмикає освітлення або активує ESP32-CAM для ідентифікації користувача. Такий підхід дозволяє оптимізувати використання енергії, адже камера працює тільки за необхідності. PIR-датчик також забезпечує комфорт користувачів, адже освітлення автоматично вмикається при вході в кімнату.

3.1.3 Світлодіодна RGB SMART-стрічка WS2812B

Світлодіодна RGB SMART-стрічка на базі WS2812B (рис. 3.3) – це сучасний елемент освітлення, який дозволяє створювати динамічні сценарії освітлення з точним налаштуванням кольорів та яскравості. Вона базується на інтегрованих драйверах у кожному світлодіоді, що забезпечує індивідуальне керування кожним пікселем.

Таблиця 3.4 – Характеристики WS2812B [14]

Параметр	**Значення**
Тип світлодіодів	RGB SMD 5050
Кількість світлодіодів	60/метр
Робоча напруга	5 В
Споживання струму	18 мА/світлодіод
Керування	Інтегрований драйвер у кожному світлодіоді
Розмір	5 мм × 5 мм на кожен піксель
Інтерфейс передачі даних	Один цифровий вхід

RGB SMART-стрічка WS2812B дозволяє індивідуально керувати кожним світлодіодом, використовуючи простий цифровий інтерфейс передачі даних. Кожен піксель отримує інформацію про колір і яскравість через єдину лінію даних.

Передача даних здійснюється за допомогою сигналу PWM (широтно-імпульсної модуляції).

Рисунок 3.3 – RGB SMART-стрічка WS2812B [14]

Щоб забезпечити стабільну роботу стрічки, в системі використовуються:

резистор 330 Ом – обмежує струм на лінії даних, захищаючи світлодіоди від перенапруги;

конденсатор JCCON 1000 мкФ 6.3 В – знижує пульсації напруги, що виникають через різке змінення яскравості стрічки. Це гарантує тривалу і стабільну роботу.

RGB SMART-стрічка забезпечує можливість динамічного налаштування освітлення. Вона дозволяє створювати індивідуальні сценарії освітлення для кожного користувача, враховуючи умови зовнішнього освітлення та його вподобання.

3.1.4 Датчика освітленості GY-302 BH1750

Датчик освітленості GY-302 BH1750 є одним із ключових елементів системи автоматизації освітлення, що відповідає за вимірювання рівня зовнішнього освітлення. Цей датчик дозволяє системі адаптувати інтенсивність штучного освітлення, щоб забезпечити комфортні умови для користувачів та зменшити енергоспоживання. Завдяки високій точності вимірювань і простоті використання, GY-302 BH1750 є оптимальним вибором для інтеграції в інтелектуальні системи освітлення.

Рисунок 3.5 – GY-302 BH1750 [15]

Таблиця 3.6 – Характеристики GY-302 BH1750

Параметр	**Значення**
Діапазон вимірювань	1–65535 люксів
Точність	±20%
Інтерфейс передачі даних	I2C
Робоча напруга	3.3–5 В
Робочий струм	0.12 мА
Розміри	18.5 мм × 13.9 мм

GY-302 BH1750 виконує функцію моніторингу зовнішнього освітлення, що дозволяє системі адаптувати інтенсивність штучного освітлення відповідно до поточних умов. Якщо рівень природного освітлення достатній, система може зменшити або повністю вимкнути штучне освітлення, заощаджуючи електроенергію. У випадку низького рівня освітленості система автоматично збільшує яскравість, забезпечуючи комфортні умови для користувачів.

3.1.5 Блок живлення 12 В, 2 А

Блок живлення є критично важливим компонентом системи, який забезпечує стабільну подачу напруги для всіх підключених елементів. Його основною функцією є перетворення змінної напруги з електричної мережі на постійну напругу, необхідну для роботи системи.

Адаптер підключається до макетної плати MB-102 через модуль живлення. Він слугує джерелом енергії для датчиків та мікроконтролера. Для забезпечення стабільної роботи світлодіодної стрічки WS2812B використовуються додаткові елементи стабілізації: конденсатор 1000 мкФ для згладжування напруги та резистор на 330 Ом для захисту лінії даних.

Блок живлення 12 В, 2 А є ключовим елементом системи, що гарантує стабільну та надійну роботу всіх компонентів. Завдяки його високій потужності та захисту від перевантажень, система може працювати без збоїв навіть при інтенсивному використанні освітлення та інших функцій. Цей компонент забезпечує базу для ефективного та безпечного функціонування всієї системи.

3.1.6 Програматор YP-05

Програматор YP-05 є необхідним елементом для завантаження прошивки в мікроконтролер ESP32-CAM. Через те, що ESP32-CAM не має вбудованого USB-інтерфейсу для прямого програмування, використання зовнішнього програматора стає обов’язковим. YP-05 забезпечує просте та надійне підключення до ESP32-CAM, дозволяючи швидко завантажувати код і налагоджувати систему.

YP-05 використовується на етапі розробки системи для завантаження коду. Завдяки цьому програматору забезпечується можливість не лише завантажувати прошивку, але й проводити налагодження системи через серійний монітор.

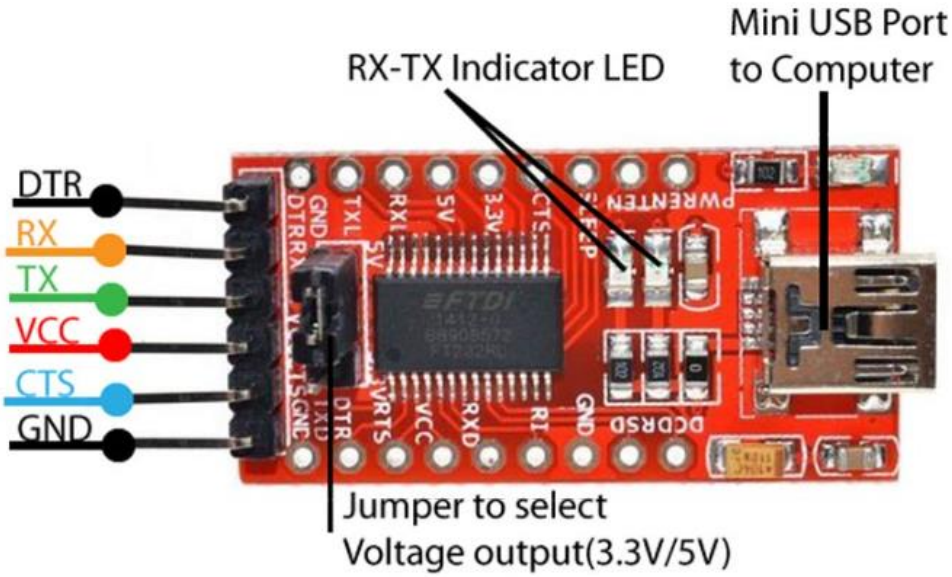


Рисунок 3.6 – YP-05 [16]

Описані вище компоненти є основою для побудови системи адаптивного управління освітленням. Кожен з них виконує унікальну функцію, забезпечуючи злагоджену роботу всієї системи. Для кращого розуміння структури пристрою на наступній схемі представлено з’єднання компонентів, їх функціональні зв’язки та розташування (рис. 3.7).

Схема дозволяє візуалізувати інтеграцію мікроконтролера, датчиків, світлодіодної стрічки та інших елементів, забезпечуючи цілісне уявлення про роботу пристрою. Такий підхід спрощує як етапи налагодження системи, так і подальшу її експлуатацію.

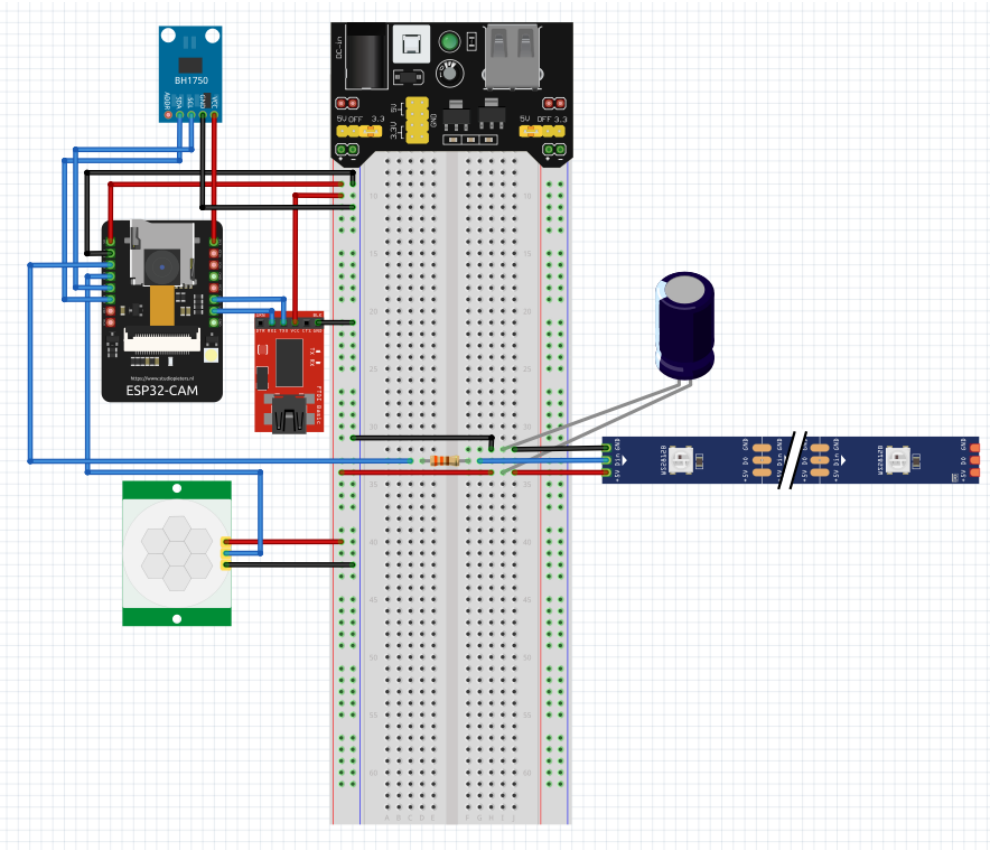
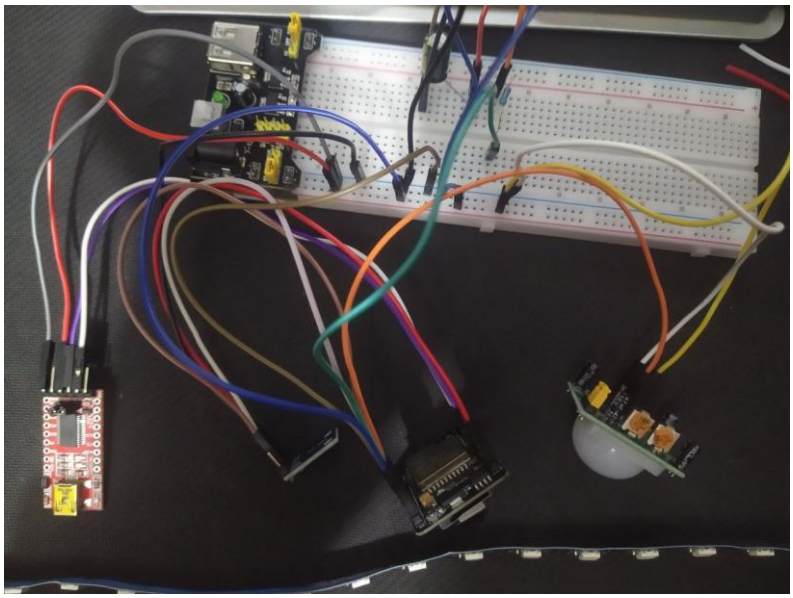
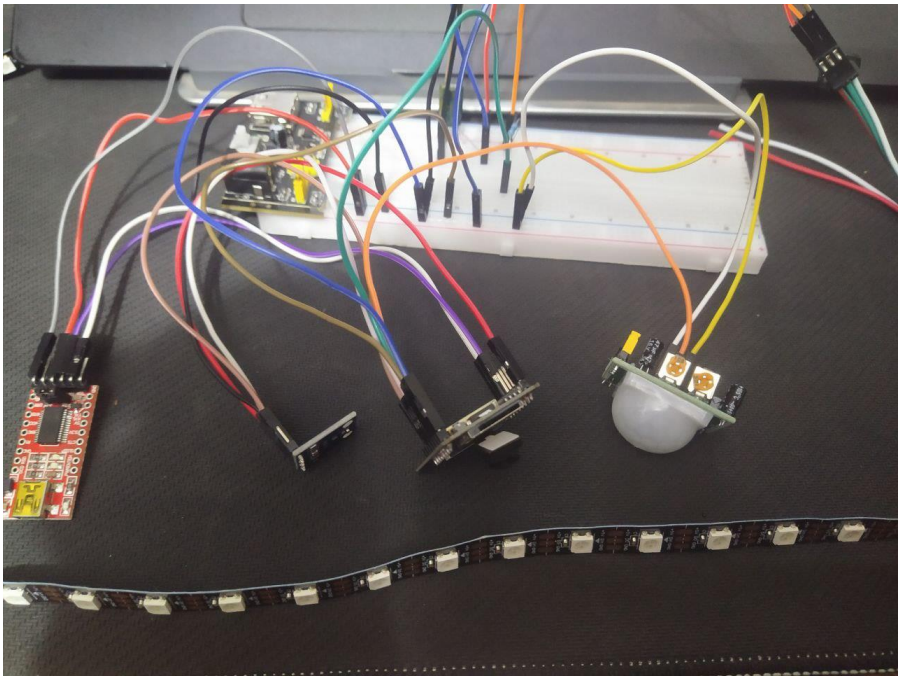


Рисунок 3.7 – Схема пристрою

Готовий результат інтеграції та функціональної взаємодії компонентів представлено на наступному рисунку (рис. 3.8).



а)



б)

Рисунок 3.8 – Готовий пристрій (а, б)

3.2 Програмне забезпечення системи

Програмне забезпечення є основою, яка забезпечує роботу системи адаптивного управління освітленням. Воно дозволяє з’єднати всі компоненти: мікроконтролер, датчики, сервер і мобільний застосунок, забезпечуючи їхню злагоджену взаємодію.

3.2.1 Пристрій керування освітленням

1) Вибір середовища та мови програмування

Для програмування пристрою було обрано Arduino IDE (рис. 3.9) як середовище розробки та C++ (Arduino Framework) як основну мову програмування. Arduino IDE забезпечує зручний інтерфейс для написання, компіляції та завантаження коду на мікроконтролер ESP32-CAM. Простота використання цього

інструменту, велика спільнота користувачів і підтримка широкого спектра бібліотек зробили його ідеальним вибором для розробки цього проекту.

Рисунок 3.9 – Arduino IDE [17]

C++ у середовищі Arduino Framework дозволяє створювати складні алгоритми, оптимізовані для виконання на мікроконтролерах, зберігаючи при цьому простоту написання коду.

2) Загальний опис та алгоритм роботи пристрою

Пристрій базується на мікроконтролері ESP32-CAM, який інтегрує всі компоненти системи. Його головним завданням є збирання даних, їх обробка та управління освітленням відповідно до змінних умов і вподобань користувачів.

Робота пристрою починається з ініціалізації всіх компонентів і встановлення з’єднання з Wi-Fi для зв’язку із сервером. PIR-датчик постійно моніторить приміщення для виявлення руху. У разі активації датчика запускається камера для розпізнавання обличь. Паралельно з цим зчитується рівень зовнішнього освітлення за допомогою BH1750. Зібрані дані передаються на сервер у форматі JSON для аналізу. Сервер опрацьовує отриману інформацію та надсилає команди пристрою

для регулювання освітлення. RGB-світлодіодна стрічка змінює яскравість і колір відповідно до команд, забезпечуючи комфортне освітлення.

Ця інтеграція програмного забезпечення дозволяє системі адаптуватися до зовнішніх умов і персоналізованих налаштувань користувачів, забезпечуючи високий рівень енергоефективності та зручності.

3) Використання бібліотеки та їхнє призначення

Програмне забезпечення пристрою використовує низку бібліотек, які забезпечують легку інтеграцію апаратних компонентів і спрощують розробку. Для роботи з датчиком освітленості BH1750 використовується бібліотека BH1750, яка дозволяє точно вимірювати рівень зовнішнього світла та аналізувати зміну зовнішніх умов. Бібліотека WiFi використовується для підключення до мережі Wi-Fi, що дає змогу пристрою обмінюватися даними із сервером.

Для роботи з HTTP-запитами застосовується бібліотека HTTPClient, яка дозволяє легко організувати передачу даних між пристроєм і сервером. Управління RGB-світлодіодною стрічкою здійснюється за допомогою бібліотеки Adafruit_NeoPixel, яка дозволяє точно регулювати яскравість і колір світлодіодів, а також реалізувати адаптивне освітлення залежно від зовнішніх умов та побажань користувача.

Для передачі даних у форматі JSON застосовується бібліотека ArduinoJson, яка спрощує формування та розбір структурованих даних, забезпечуючи швидкий і надійний обмін інформацією між компонентами системи. Цей набір бібліотек забезпечує ефективну та стабільну роботу програмного забезпечення, реалізуючи всі необхідні функції для роботи системи адаптивного освітлення.

Повний код пристрою можна переглянути в Додатку А.1.

3.2.2 Сервер

1) Вибір середовища та мови програмування

Для створення серверної частини системи адаптивного освітлення було обрано мову програмування Python та середовище розробки Visual Studio Code.

Python є оптимальним вибором для серверних додатків завдяки своїй простоті, гнучкості та широкому набору бібліотек для роботи з вебзапитами, обробки даних, базами даних та реалізації алгоритмів машинного навчання.

Використання Visual Studio Code, як інтегрованого середовища розробки, дозволяє ефективно працювати з Python завдяки вбудованій підтримці розширень для автодоповнення коду, налагодження та управління бібліотеками.

2) Вебфреймворк для реалізації серверної частини

Flask було обрано як основний вебфреймворк для реалізації серверної частини з кількох важливих причин. Це легкий і модульний інструмент, який забезпечує швидку розробку RESTful API. Flask дозволяє створювати маршрути для обробки HTTP-запитів, легко інтегрувати сторонні бібліотеки та розширення, а також масштабувати серверну частину за потреби. Фреймворк є ідеальним для проєктів типу IoT завдяки своїй простій структурі та відсутності зайвих залежностей, що дозволяє зосередитися саме на реалізації функціональних можливостей системи [18]

Flask забезпечує високу продуктивність при виконанні невеликих, але частих запитів, що критично для IoT-проєктів, де пристрої постійно взаємодіють із сервером. Його простота у використанні дозволяє легко налаштувати логіку обробки даних і інтегрувати функції, такі як розпізнавання облич через бібліотеку DeepFace.

Рисунок 3.10 – Flask [18]

3) Загальний опис та алгоритм роботи серверу

Сервер виконує ключову роль у взаємодії між пристроями, мобільним застосунком та базою даних. Його завдання включають прийом даних із пристроїв, їх обробку, збереження в базі даних і передачу відповідей на основі алгоритмів адаптивного освітлення. Сервер забезпечує персоналізацію, ідентифікацію користувачів та динамічну зміну параметрів освітлення відповідно до їхніх потреб і зовнішніх умов.

Основний алгоритм роботи серверу розпочинається з прийому даних від мобільного застосунку. Застосунок передає побажання користувача щодо освітлення (наприклад, рівень яскравості або колірний режим), а також зображення обличчя користувача для ідентифікації. Паралельно пристрій, активований сигналом датчика руху, надсилає фото для порівняння.

Сервер отримує ці дані та виконує розпізнавання облич за допомогою бібліотеки DeepFace. Результат розпізнавання використовується для пошуку відповідного запису в базі даних, де зберігаються налаштування користувача. Якщо зображення користувача збігається із записом у базі даних, сервер виконує перевірку отриманих побажань з налаштуваннями в базі, комбінуючи їх із зовнішніми умовами, такими як рівень природного освітлення.

На основі аналізу даних сервер формує відповідні команди для пристрою. Ці команди включають інформацію про необхідний рівень освітлення, колірний

режим або сценарій роботи. Команди надсилаються на пристрій, який виконує коригування освітлення відповідно до поточних умов і налаштувань користувача.

Цей алгоритм забезпечує персоналізоване управління освітленням, враховуючи побажання користувача, зовнішні умови та історію взаємодії системи.

4) Використання бібліотек та їхнє призначення

Для реалізації серверної частини використовуються такі бібліотеки:

– Flask: Основний вебфреймворк для обробки HTTP-запитів, маршрутизації та створення RESTful API;

– Flask-SQLAlchemy: Забезпечує зручний доступ до бази даних SQLite для збереження налаштувань користувачів;

– DeepFace: це сучасна бібліотека Python для розпізнавання облич, яка забезпечує високий рівень точності завдяки використанню глибоких нейронних мереж. Вона підтримує кілька відомих моделей для розпізнавання, таких як VGG-Face, Google FaceNet, OpenFace, DeepID, ArcFace та Dlib [19];

– os: Для роботи з файлами та обробки фото, збережених на сервері;

– base64: Використовується для передачі та обробки зображень, які надсилаються у вигляді рядків коду.

Повний код серверної частини можна переглянути в Додатку А.2.

3.2.3 Мобільний застосунок

1) Вибір середовища та мови програмування

Мобільний застосунок розроблено на основі Flutter, який є популярним інструментом для створення багатоплатформних додатків.

Flutter – це фреймворк з відкритим кодом, розроблений Google, який дозволяє створювати високоякісні багатоплатформні мобільні застосунки з єдиною базою коду. Він використовує мову програмування Dart, яка відома своєю простотою, високою продуктивністю та оптимізацією для мобільних платформ. Однією з

ключових переваг Flutter є можливість створення застосунків, які виглядають і працюють природно як на Android, так і на iOS.

Середовищем було обране Visual Studio Code.

2) Загальний опис роботи застосунку та використання бібліотек

Мобільний застосунок для управління системою адаптивного освітлення побудований на базі Flutter і забезпечує користувачам зручний інструмент для персоналізації освітлення в приміщенні. Програма починає свою роботу з екранів onboarding, які знайомлять користувача з ключовими функціями системи, такими як персоналізація освітлення, автоматична адаптація до зовнішніх умов і можливість віддаленого керування. Ці екрани реалізовані за допомогою бібліотеки smooth_page_indicator, що додає плавну та інтуїтивну навігацію.

Після реєстрації або авторизації користувача застосунок дозволяє завантажувати фотографію для розпізнавання облич та налаштовувати вподобання щодо рівня яскравості, кольорової температури чи специфічних сценаріїв освітлення. Використання бібліотеки image_picker забезпечує доступ до галереї або камери для додавання зображення.

Програма синхронізується з серверною частиною, використовуючи бібліотеку http, яка забезпечує швидку і стабільну передачу даних. Сервер, отримуючи побажання користувача, обробляє їх і надсилає відповідні команди на пристрої. Flutter Riverpod використовується для управління станом застосунку, що гарантує плавну роботу навіть при зміні екрану чи виконанні довготривалих запитів.

У процесі використання застосунок дозволяє переглядати поточний стан освітлення, оновлювати вподобання та взаємодіяти із системою в реальному часі. Завдяки підтримці віджетів Material Design, інтерфейс виглядає сучасно та зручно для користувачів.

Повний код мобільного застосунку можна переглянути в Додатках А.3-А.7.

Висновок до розділу 3

У третьому розділі дипломної роботи було проведено детальний опис розробки апаратної та програмної частин системи адаптивного освітлення. Вибір компонентів системи, таких як ESP32-CAM, PIR-датчик, датчик освітленості BH1750 та RGB-світлодіодна стрічка, було обґрунтовано з огляду на їх функціональність, сумісність та енергозбереження. Описано архітектуру системи, яка забезпечує інтеграцію апаратних елементів та їх взаємодію через центральний мікроконтролер.

Програмна частина, розроблена для пристрою, серверу та мобільного застосунку, демонструє використання сучасних технологій та бібліотек, таких як Adafruit_NeoPixel, Flask, Flutter та Riverpod. Було описано алгоритми роботи пристрою та серверної частини, які дозволяють ідентифікувати користувачів, враховувати їхні вподобання щодо освітлення та динамічно змінювати параметри освітлення залежно від зовнішніх умов.

Мобільний застосунок, створений на платформі Flutter, забезпечує інтуїтивний інтерфейс для взаємодії користувачів із системою. Він дозволяє персоналізувати налаштування освітлення, переглядати поточний стан системи та отримувати сповіщення про її роботу.

Результатом роботи цього розділу є повноцінно реалізована система, що інтегрує апаратну частину, сервер і мобільний застосунок, забезпечуючи ефективне та персоналізоване управління освітленням у приміщенні. Це підтверджує можливість використання даного рішення в реальних умовах та його перспективність для впровадження в розумних будинках і офісах.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Мета та завдання тестування

Метою тестування є перевірка працездатності розробленої системи адаптивного управління освітленням, забезпечення її відповідності функціональним і нефункціональним вимогам, а також оцінка ефективності компонентів у реальних умовах. Тестування повинно продемонструвати, як окремі компоненти системи (мобільний застосунок, серверна частина та апаратна частина) працюють у комплексі для досягнення поставлених цілей.

Основними завданнями тестування є:

– оцінка функціональності кожного компоненту окремо;

– перевірка взаємодії між компонентами системи;

– підготовка рекомендацій для вдосконалення системи на основі отриманих результатів.

Це тестування є критичним етапом для забезпечення високої якості системи та її відповідності заявленим вимогам.

4.2 Опис методики тестування

Для перевірки роботи системи адаптивного управління освітленням було застосовано сценарний підхід, який охоплює повний цикл роботи окремих функцій. Кожен сценарій тестує взаємодію між мобільним застосунком, сервером і апаратною частиною. Це дозволяє оцінити систему як у межах окремих компонентів, так і в контексті їхньої інтеграції.

Сценарії тестування:

1) **сценарій 1: додавання нового користувача:**

а) мобільний застосунок:

– користувач заповнює форму для створення профілю та надсилає її через UI;

– очікуваний результат: форма правильно заповнюється, дані коректно передаються на сервер;

б) сервер:

– сервер обробляє запит на створення нового користувача, додає його до бази даних;

– очікуваний результат: запис успішно зберігається в базі, сервер повертає відповідь «успіх»;

2) **сценарій 2: редагування профілю користувача:**

а) мобільний застосунок:

– користувач змінює вподобання освітлення через UI;

– очікуваний результат: зміни правильно передаються на сервер;

б) сервер:

– сервер приймає запит на оновлення даних, коректно змінює їх у базі даних;

– очікуваний результат: сервер повертає успішну відповідь, зміни відображаються в базі даних;

3) **сценарій 3: видалення користувача:**

а) мобільний застосунок:

– користувач видаляє свій профіль через UI;

– очікуваний результат: запит успішно надсилається на сервер;

б) сервер:

– сервер обробляє запит, видаляє відповідний запис із бази даних;

– очікуваний результат: запис видаляється з бази, сервер повертає статус успіху;

4) сценарій 4: зміна режиму роботи:

а) мобільний застосунок:

– користувач перемикає режим з пріоритетного на компромісний;

– очікуваний результат: режим змінюється в UI;

б) сервер:

– сервер обробляє запит, оновлює режим у базі даних;

– очікуваний результат: зміна коректно зберігається;

5) сценарій 5: зміна пріоритетів:

а) мобільний застосунок:

– користувач змінює пріоритети;

– очікуваний результат: пріоритети змінюється в UI;

б) сервер:

– сервер обробляє запит, оновлює пріоритети у базі даних;

– очікуваний результат: зміна коректно зберігається;

6) сценарій 6: отримання рівня освітлення:

а) апаратна частина:

– датчик надсилає рівень природного освітлення на сервер;

– камера передає фотографію користувача для ідентифікації;

– очікуваний результат: сервер приймає дані;

б) сервер:

– сервер аналізує отримані дані, визначає потрібний рівень штучного освітлення та надсилає його пристрою;

– очікуваний результат: сервер генерує коректну відповідь.

Апаратна частина:

– пристрій отримує команду на зміну рівня освітлення;

– очікуваний результат: пристрій змінює освітлення відповідно до отриманих даних.

Методи перевірки:

– скріншоти роботи мобільного застосунку;

– логування на сервері;

– відповіді серверу;

– Записи у базі даних до і після операцій.

Сценарний підхід забезпечує повне охоплення функціональних можливостей системи, дозволяючи оцінити її продуктивність, коректність передачі даних між мобільним застосунком, сервером і апаратною частиною, а також відповідність роботи системи поставленим завданням. Використання різноманітних методів перевірки, таких як аналіз логів, перевірка записів у базі даних та скріншоти з мобільного застосунку, гарантує точність оцінки і виявлення можливих недоліків на кожному етапі роботи системи.

4.3 Проведення тестування

4.3.1 Сценарій 1

Процес тестування сценарію додавання нового користувача розпочався із взаємодії через мобільний застосунок. Після авторизації та переходу на головний екран (рис. 4.1), необхідно натиснути кнопку «Додати нового користувача».

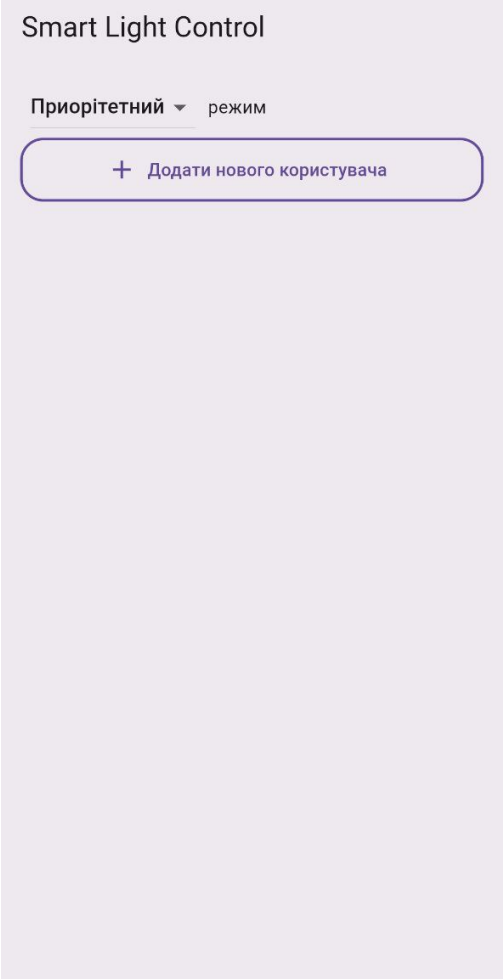


Рисунок 4.1 – Головний екран застосунку

Після цього відкривається екран для налаштування параметрів освітлення (рис. 4.2). Заповнивши всі необхідні поля та натиснувши кнопку «Зберегти налаштування», у консолі відобразиться відповідне повідомлення:

```
LightingPreferences (id: a6d6b3aa-2b88-4c3a-ad67-fe87f5d7b086, name: Leo, photo: File: /data/user/0/com.example.nasters_project/cache/scaled_d139d173-af20-4a3e-b24e-665173d2dc6c7929231939418132713.jpg', bri ghtness: 75.0, colorMode: warm, autoColorMode: false, warnLightStartTime: TimeOfDay (18:00), lightingTimeout: 5.0, adaptiveLevel: 60.0, priority: 1)
```

Це свідчить про успішну передачу даних на сервер для подальшої обробки.

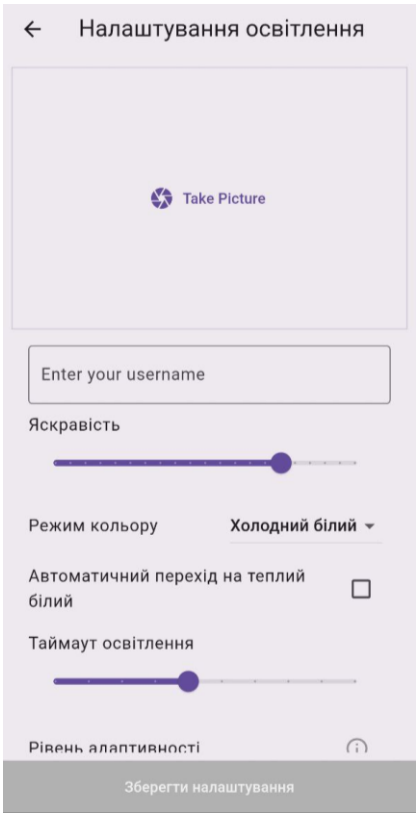


Рисунок 4.2 – Екран «Налаштування освітлення»

Після отримання даних сервер зберігає інформацію у базі даних (рис. 4.3) та надсилає застосунку відповідь із статусом виконання запиту.

```
“POST /preferences HTTP/1.1” 200 -
```

	user_id	name	brightness	color_mode	auto_color_mode	warm_light_start_time	photo_path
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	a6d6b3aa-2b88-4c3a-ad67-fe87f5d7b086	Leo	75.0	warm	0	18:00	uploads/a6d6b3aa-2b88-4c3a-ad67-...

Рисунок 4.3 – Доданий запис в базу даних

Після успішного додавання новий користувач відобразиться на головному екрані (рис. 4.4), де стане доступним для подальшої взаємодії.

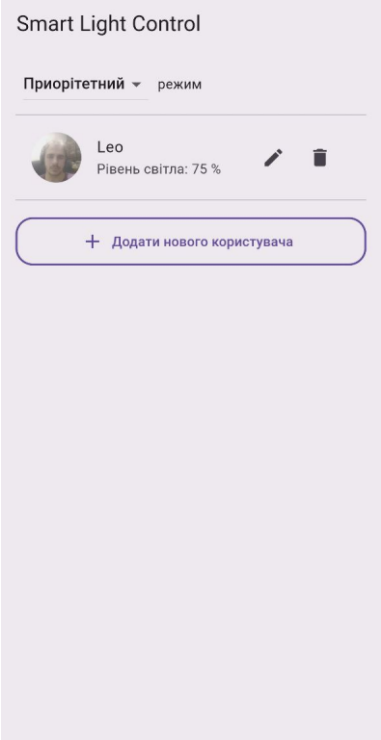


Рисунок 4.4 – Успішно доданий користувач

Отже, можна дійти висновку, що функціонал додавання побажань користувача працює належним чином і виконується без помилок.

4.3.2 Сценарій 2

Для зміни інформації користувача необхідно натиснути на іконку редагування. Після цього відкриється екран редагування налаштувань існуючого користувача (рис. 4.5), де можна змінити фото користувача та всі параметри, пов’язані з налаштуванням освітлення. Після внесення змін та натискання кнопки «Зберегти налаштування» оновлені дані надсилаються на сервер. У консолі сервера відображається повідомлення про успішну обробку запиту.

```
"POST /preferences/a6d6b3aa-2b88-4c3a-ad67-fe87f5d7b086 HTTP/1.1" 200 –
```

Також у застосунок надходить повідомлення про успішне оновлення даних користувача.

```
"message":"Preferences updated successfully"
```

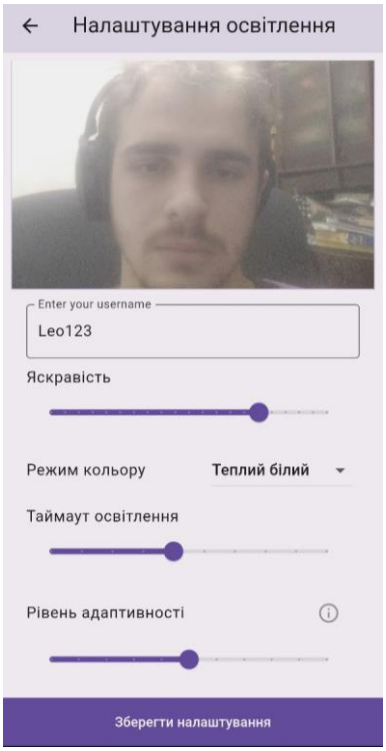
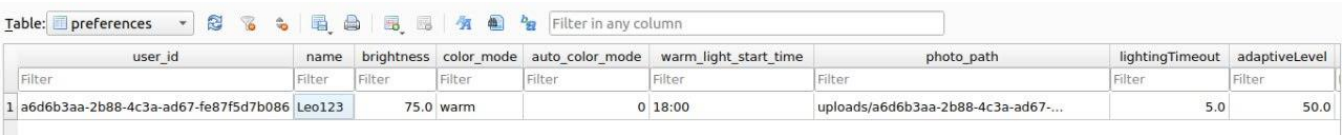


Рисунок 4.5 – Редагування налаштувань існуючого користувача

Оновлені дані успішно збережено у базі даних (рис. 4.6), замінивши попередні (рис. 4.3).



	user_id	name	brightness	color_mode	auto_color_mode	warm_light_start_time	photo_path	lightingTimeout	adaptiveLevel
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	a6d6b3aa-2b88-4c3a-ad67-fe87f5d7b086	Leo123	75.0	warm	0	18:00	uploads/a6d6b3aa-2b88-4c3a-ad67-...	5.0	50.0

Рисунок 4.6 – Оновлені дані користувача

Оновлені дані також відображаються на головному екрані застосунку (рис. 4.7), замінюючи попередні, які були показані (рис. 4.4).

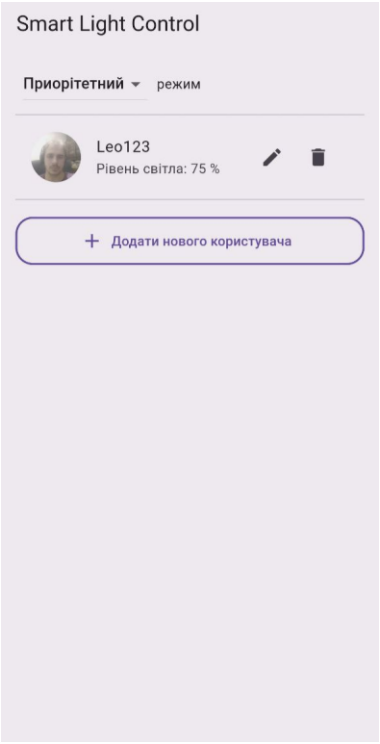


Рисунок 4.7 – Змінені дані користувача

За результатами проведеного тестування можна зробити висновок, що функціонал оновлення побажань користувача працює коректно та без помилок.

4.3.3 Сценарій 3

Для видалення користувача необхідно натиснути на іконку видалення. Після цього з’явиться спливаюче вікно з підтвердженням дії (рис. 4.8). У разі натискання кнопки «Так», користувача буде видалено з бази даних.

На сервері відобразиться повідомлення про успішну обробку запиту:

```
"DELETE /preferences/a6d6b3aa-2b88-4c3a-ad67-fe87f5d7b086 HTTP/1.1" 200 –
```

У консолі застосунку відобразиться повідомлення про успішне видалення користувача:

```
Preferences deleted successfully: {"message":"Preference with id a6d6b3aa-2b88-4c3a-ad67-fe87f5d7b086 deleted successfully"}
```

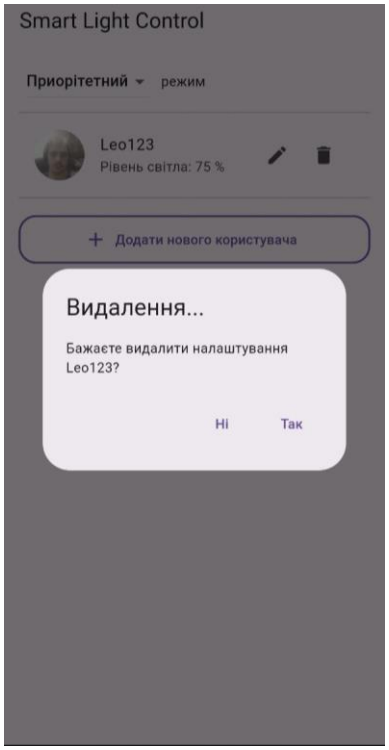


Рисунок 4.8 – Спливаюче вікно для підтвердження видалення налаштувань користувача

Успішне видалення можна підтвердити, перевіривши базу даних, де видалений користувач більше не буде присутній (рис. 4.9).

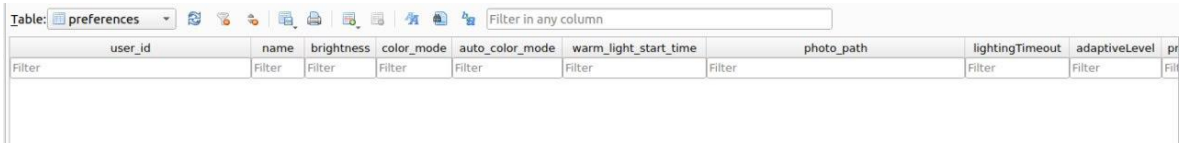


Рисунок 4.9 – Пуста база даних після видалення користувача

За результатами проведеного тестування видалення налаштувань користувача можна зробити висновок, що функціонал працює коректно та виконує всі необхідні дії без помилок.

4.3.4 Сценарій 4

Для зміни режиму роботи використовується випадаюче меню (рис. 4.1). Після вибору нового значення в цьому меню, відправляється запит на сервер, який обробляє зміну режиму. Результат обробки запиту можна переглянути в консолі:

```
"POST /mode HTTP/1.1" 200 –
```

У консолі застосунку з’явиться повідомлення про успішну зміну режиму:

```
Settings updated successfully on the server
```

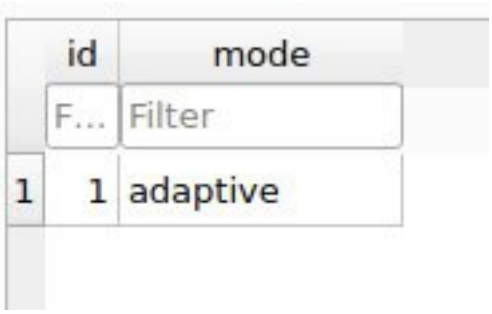
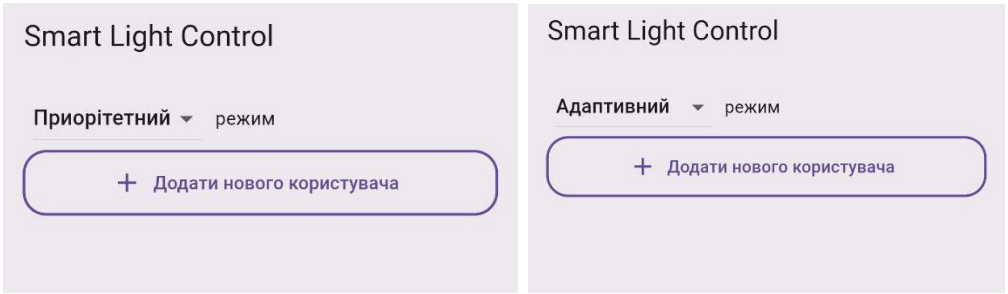


Рисунок 4.10 – Оновлене значення режиму в базі даних



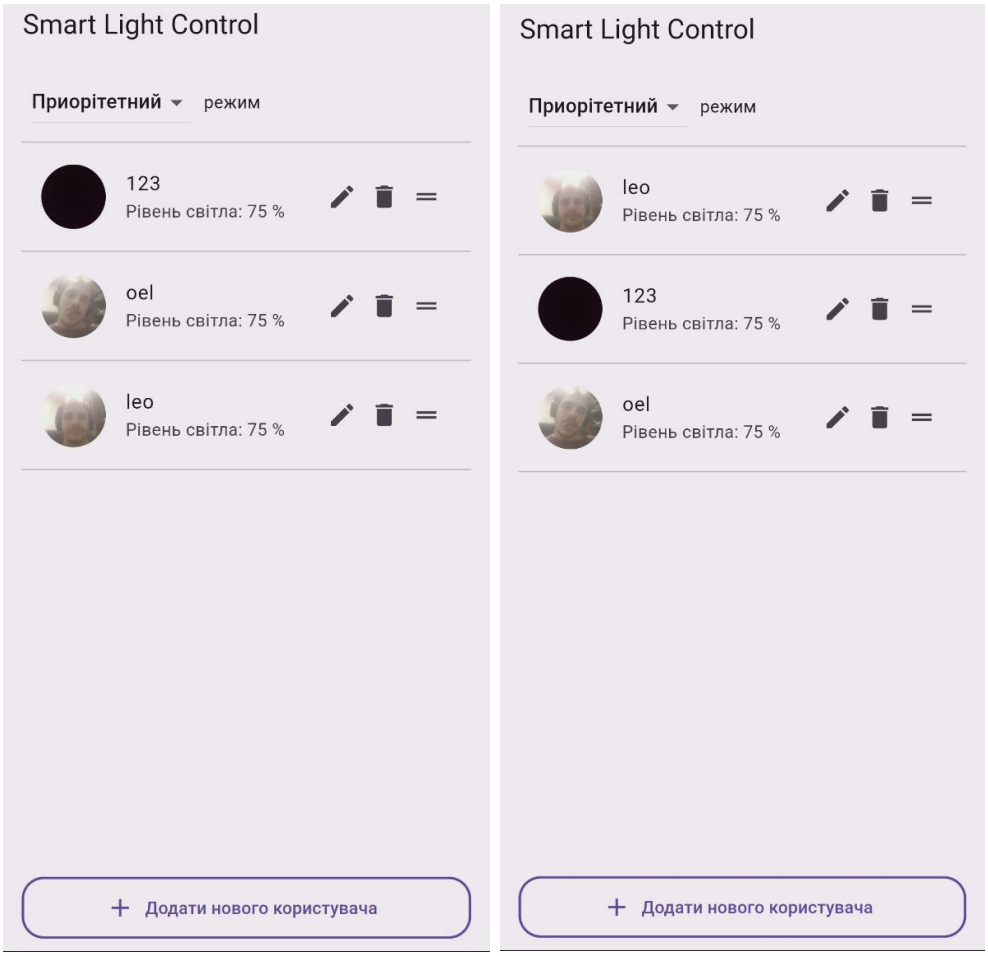
а) б)

Рисунок 4.11 – Режим роботи: а) значення до зміни, б) значення після зміни

За результатами проведеного тестування можна зробити висновок, що функціонал зміни режиму роботи працює коректно та виконує всі необхідні дії без помилок.

4.3.5 Сценарій 5

Для тестування зміни пріоритетів необхідно перейти в пріоритетний режим. Після цього можна змінювати порядок налаштувань, утримуючи іконку перетягування (рис. 4.12 а) та перетягуючи елемент на бажану позицію. Зміна пріоритетів відбувається відповідно до нової позиції в списку (рис. 4.12 б).



а) б)

Рисунок 4.12 – Зміна пріоритетів користувачів: а) розташування перед зміною, б) розташування після зміни

Після зміни пріоритетів відправляється запит на сервер для обробки змін. У консолі сервера можна побачити підтвердження обробки запиту:

```
"POST /update_priorities HTTP/1.1" 200 -
```

У консолі застосунку також з’являється повідомлення про успішну зміну пріоритетів:

```
Priorities updated successfully
```

Table: preferences

user_id	name	brightness	color_mode	auto_color_mode	warm_light_start_time	photo_path	lightingTimeout	adaptiveLevel	priority
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1 9-2233-413b-a566-...	leo	75.0	cool	0	18:00	uploads/7f48aeb9-2233-413b-a566-...	5.0	50.0	3
2 3-17a7-40f5-8b02-5c13d0f1e6bd	oel	75.0	cool	0	18:00	uploads/...	5.0	50.0	2
3 a0-0a1f-49bf-ab74-b31d7f4f19ee	123	75.0	cool	0	18:00	uploads/0d7599a0-0a1f-49bf-ab74-...	5.0	50.0	1

а)

Table: preferences

user_id	name	brightness	color_mode	auto_color_mode	warm_light_start_time	photo_path	lightingTimeout	adaptiveLevel	priority
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1 9-2233-413b-a566-...	leo	75.0	cool	0	18:00	uploads/7f48aeb9-2233-413b-a566-...	5.0	50.0	1
2 3-17a7-40f5-8b02-5c13d0f1e6bd	oel	75.0	cool	0	18:00	uploads/...	5.0	50.0	3
3 a0-0a1f-49bf-ab74-b31d7f4f19ee	123	75.0	cool	0	18:00	uploads/0d7599a0-0a1f-49bf-ab74-...	5.0	50.0	2

б)

Рисунок 4.13 – Зміна значень в базі даних: а) до зміни, б) після зміни

За результатами проведеного тестування можна зробити висновок, що функціонал зміни пріоритетів працює коректно та забезпечує належне виконання всіх необхідних дій без помилок.

4.3.6 Сценарій 6

Для тестування обробки даних з пристрою, таких як фотографія та рівень освітлення в кімнаті, необхідно запустити сервер і під'єднати пристрій до мережі.

Після виявлення руху пристрій автоматично робить фото і надсилає його на сервер разом із даними про рівень освітлення, які вимірюються за допомогою датчика (рис. 4.14).

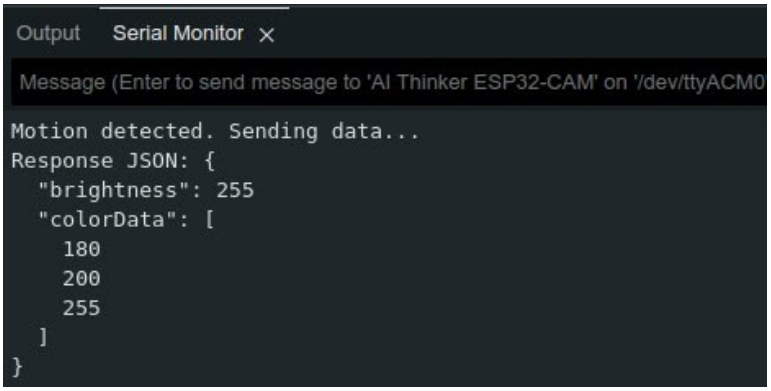


Рисунок 4.14 – Відправлення даних на сервер та отримання відповіді

Після отримання даних сервер розпочинає обробку фотографії (рис. 4.15).

Рисунок 4.15 – Обробка даних сервером

З логів сервера видно, що було успішно виявлено користувача Leo, і застосовано адаптивний режим. У цьому режимі, якщо в кімнаті перебуває лише один користувач, використовуються всі налаштування, прив'язані до розпізнаного користувача.

Якщо змінити налаштування через мобільний застосунок, то під час наступного запиту пристрій врахує ці зміни, і результати будуть оновлені відповідно до нових налаштувань (рис. 4.17).

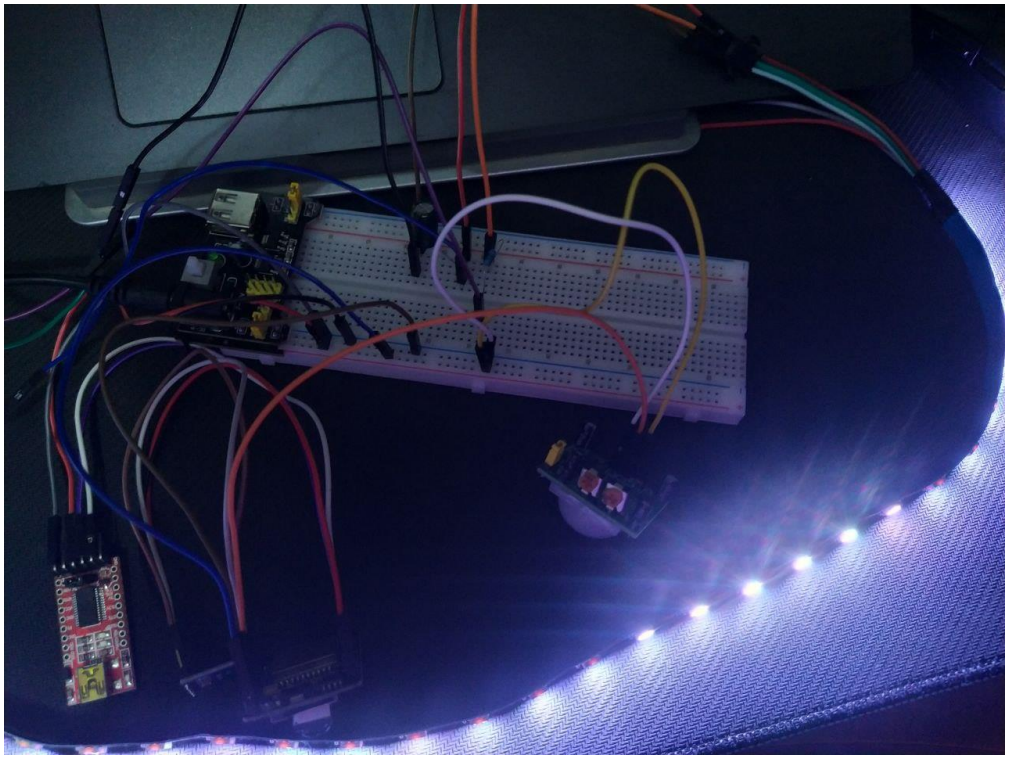


Рисунок 4.16 – Увімкнення освітлення на основі побажань користувача Leo

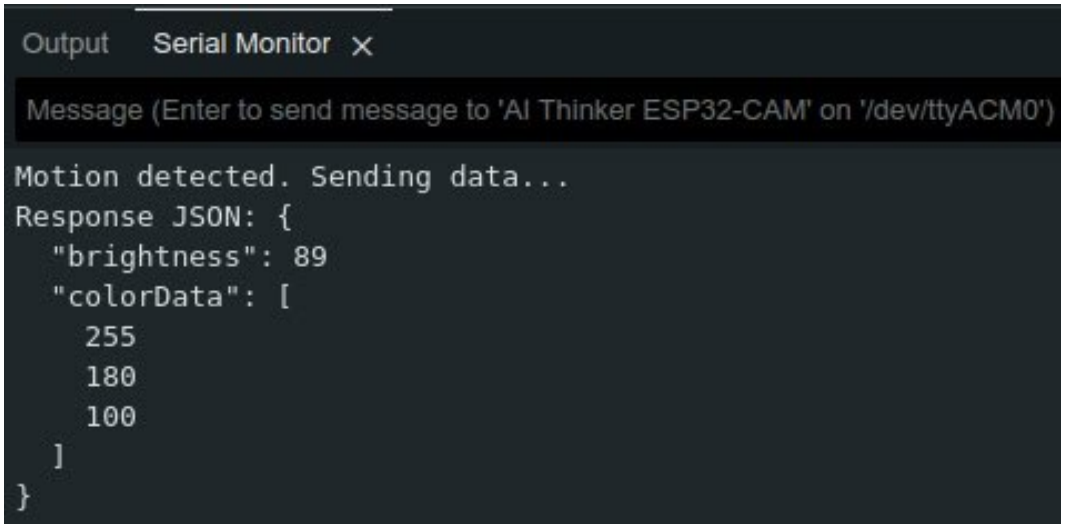


Рисунок 4.17 – Отримання оновлених даних

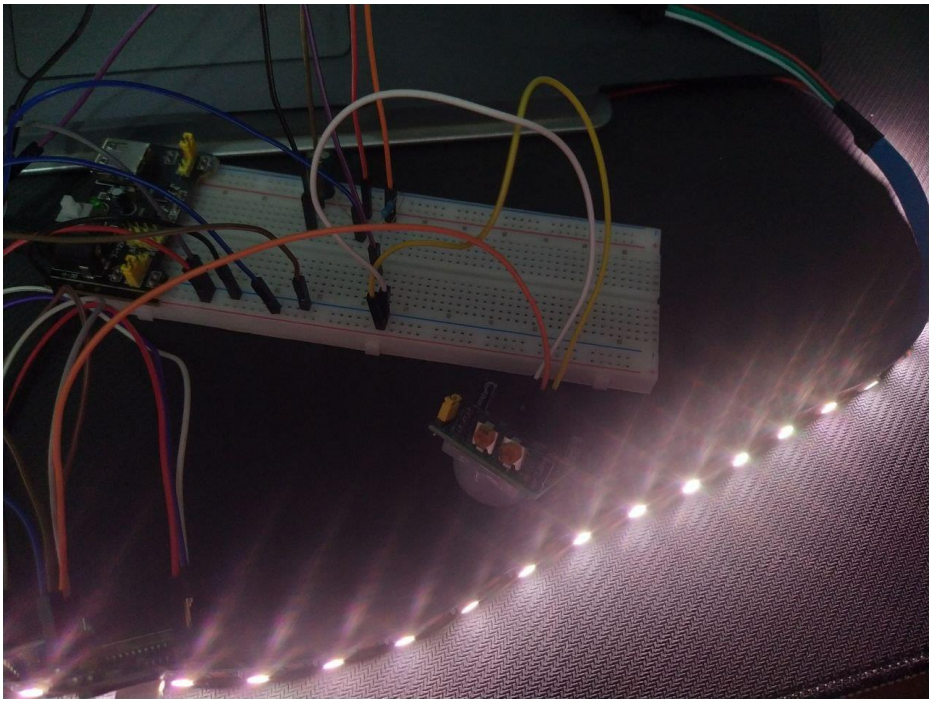


Рисунок 4.18 – Увімкнення системи після зміни налаштувань
в мобільному застосунку

Також було проведено тестування можливості обробки фотографій, на яких присутні два або більше користувачів.

```
24-12-09 19:34:56 - Searching temp_photo/temp_photo.jpg in 2 length datastore
24-12-09 19:34:56 - find function duration 0.47050976753234863 seconds
Detected Ryan
Adaptive mode
Settings of Leo, Ryan are used
Sending to ESP32-CAM: {'brightness': 189, 'colorData': [180, 200, 255]}
127.0.0.1 - - [09/Dec/2024 19:34:56] "POST /process_data HTTP/1.1" 200 -
```

Рисунок 4.19 – Обробка зображення з двома користувачами в адаптивному режимі

Як видно з логів серверної частини, обробка зображення була виконана успішно, після чого було здійснено розрахунок освітлення та відправлено відповідні дані на пристрій.

Лог сервера підтверджує успішну обробку запиту:

```
"POST /mode HTTP/1.1" 200 –
```

Після зміни режиму на пріоритетний, у ситуації, коли в кімнаті знаходяться два користувачі, система переключилася на налаштування користувача з найвищим пріоритетом. У цьому випадку пріоритет був відданий користувачу Leo (рис. 4.20).

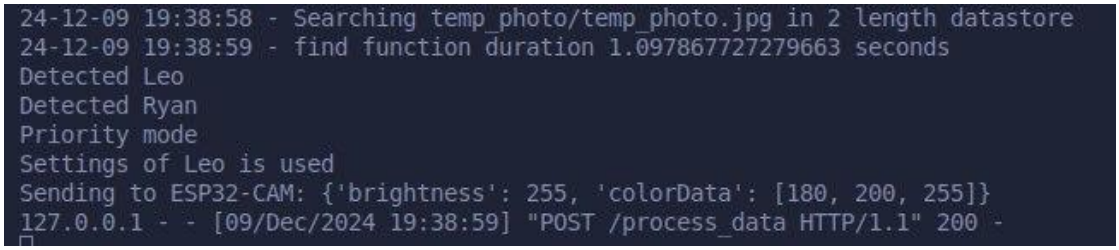


Рисунок 4.20 – Використання приоритетного режиму при 2 користувачах

Результат, відправлений на сервер у цьому випадку, буде аналогічним тому, що зображений на рисунку 4.15.

4.4 Оцінка енергоефективності системи

Розроблена система освітлення демонструє високий рівень енергоефективності завдяки адаптивному підходу, який враховує вподобання користувачів, природне освітлення та режими роботи світлодіодної стрічки (LED). Для оцінки її ефективності було виконано розрахунки споживання енергії в залежності від сценаріїв використання, а також порівняння з традиційними системами освітлення.

4.4.1 Традиційні системами

Традиційні системи освітлення, які працюють за принципом "включити-виключити", не враховують природне освітлення або вподобання користувачів. Вони забезпечують постійний рівень освітлення, незалежно від потреб, що призводить до значного перевитрачання енергії.

Наприклад, лампа потужністю 18 Вт, яка працює 6 годин на день, споживає:

$$E_{trad} = 18\text{Вт} \times 6\ \text{год} = 108\ \text{Вт год/день}$$

При відсутності функцій автоматичного вимкнення та адаптації це перевищує реальні потреби, особливо вдень, коли є природне освітлення.

4.4.2 Системами з фіксованими правилами

Сучасні системи з фіксованими правилами, такі як Philips Hue, Lutron або Caseta, оптимізують використання енергії за рахунок жорстко запрограмованих сценаріїв роботи:

Лампа Philips Hue з потужністю 10 Вт у активному режимі (6 годин) і 0.5 Вт у режимі очікування (18 годин) споживає:

$$E_{\text{Philips Hue}} = (10\text{Вт} \times 6\text{год}) + (0.5\text{Вт} \times 18\text{год}) = 69\text{Вт год/день}.$$

Датчики (наприклад, руху або освітленості) додають до загального споживання:

$$E_{\text{sensors}} = 0.2\text{Вт} \times 24\text{год} = 4.8\text{Вт год/день}.$$

Загальне споживання системи:

$$E_{\text{Philips Hue total}} = 69 + 4.8 = 73.8\text{Вт год/день}.$$

Ці системи показують вищий рівень енергоефективності у порівнянні з традиційними лампами за рахунок оптимізації роботи, але їх фіксовані сценарії не завжди враховують реальні потреби або рівень природного освітлення.

4.4.3 Розроблювана система

Розроблена система освітлення використовує адаптивний підхід, враховуючи:

– поточний рівень природного освітлення;
– вподобання кількох користувачів одночасно;
– динамічне коригування яскравості та кольору світла.

Результати розрахунків:

– мінімальна яскравість (2 години): 3.6 Вт·год;

– середня яскравість (2 години): 18.0 Вт·год;
– максимальна яскравість (2 години): 36.0 Вт·год;
– режим очікування (18 годин): 5.4 Вт·год.

Загальне споживання LED-стрічки:

$$E_{\text{LED}} = 3.6 + 18.0 + 36.0 + 5.4 = 63.0\text{Вт год/день}$$

Інші компоненти системи (ESP32-CAM, датчики): 7.93 Вт·год/день.

Сумарне споживання розробленої системи:

$$E = 63.0 + 7.93 = 70.93\text{Вт год/день.}$$

Розроблена система споживає приблизно на 34.3 % менше енергії, ніж традиційна лампа потужністю 18 Вт, яка працює за принципом «включити-виключити». У порівнянні з системами, які використовують фіксовані правила, наприклад, Philips Hue, наша система демонструє на 4.5 % нижче споживання. Це пояснюється адаптивним підходом до освітлення, що враховує рівень природного освітлення, вподобання користувачів та поточні умови в приміщенні. Завдяки цьому наша система забезпечує більшу гнучкість і економію енергії, водночас зберігаючи високий рівень комфорту для користувачів.

4.5 Висновок до розділу 4

У процесі тестування системи було перевірено всі ключові сценарії її роботи, що охоплювали додавання нового користувача, редагування його налаштувань, інтеграцію із серверною частиною, передачу та обробку даних, а також динамічне регулювання параметрів освітлення. Кожен сценарій було ретельно протестовано, включно з перевіркою коректності взаємодії між мобільним додатком, сервером і апаратною частиною. Результати тестування підтвердили відповідність системи функціональним вимогам і її здатність працювати у реальних умовах.

Аналіз показав, що наша система, завдяки адаптивному підходу, споживає менше енергії порівняно як із традиційними системами освітлення, так і з

сучасними комерційними системами з фіксованими правилами, такими як Philips Hue. Ефективність досягається за рахунок врахування рівня природного освітлення, уподобань користувачів та динамічного регулювання параметрів освітлення.

Тестування підтвердило, що система забезпечує баланс між енергоефективністю та комфортом користувачів. Наприклад, навіть у сценаріях з максимальним навантаженням, вона демонструє оптимальне споживання електроенергії, що є важливим фактором для сучасних екологічних рішень.

Таким чином, результати тестування доводять, що розроблена система відповідає заданим технічним та експлуатаційним вимогам, демонструючи високий рівень функціональності та конкурентну енергоефективність.

ВИСНОВКИ

Виконана робота досягла поставленої мети – створення адаптивної системи управління освітленням, яка враховує природне освітлення, вподобання користувачів та забезпечує високий рівень енергоефективності. У процесі роботи було проведено аналіз сучасних рішень, розроблено архітектуру системи, виконано її реалізацію та тестування у реальних умовах. Результати підтвердили коректність функціонування системи та відповідність її основним технічним і експлуатаційним вимогам.

Система демонструє стабільну роботу, забезпечує динамічне регулювання параметрів освітлення залежно від змінних умов, і відзначається зниженим енергоспоживанням. Тестування підтвердило, що розроблена система споживає менше електроенергії, ніж традиційні системи освітлення та сучасні комерційні рішення, такі як Philips Hue або Lutron. Це досягається завдяки адаптивним алгоритмам, які оптимізують яскравість та вибір кольору освітлення з урахуванням індивідуальних потреб користувачів.

Однією з ключових переваг системи є її гнучкість і масштабованість. Вона здатна взаємодіяти з кількома користувачами одночасно та забезпечувати енергозбереження без втрати комфорту. Адаптивний підхід до управління освітленням дозволяє не лише підвищити ефективність споживання енергії, але й створює комфортне середовище для користувачів у різних сценаріях використання.

Розроблена система має значний потенціал для впровадження у житлових, офісних та комерційних приміщеннях. Перспективним напрямком її розвитку є інтеграція із зовнішніми IoT-рішеннями, вдосконалення алгоритмів адаптації для складніших сценаріїв, а також розширення функціональності за рахунок додавання нових сенсорів та модулів. Це дозволить ще більше підвищити енергоефективність системи та її конкурентоспроможність.

Таким чином, виконана робота сприяє розвитку сучасних систем управління освітленням, пропонуючи ефективний інструмент для покращення енергоефективності та комфорту користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Козявко Л. О., Пузирьов С. В. Адаптивне управління освітленням в приміщенні за допомогою датчиків та алгоритмів машинного навчання. *Ольвійський форум-2024: стратегії країн причорноморського регіону в геополітичному просторі*. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2024.С 159-161.

2. Shaikh S. A. Comparative Analysis of Different Commercial Lights. *Sukkur IBA Journal of Emerging Technologies*. 2018. Vol. 1, no. 1. P. 34–44. DOI:10.30537/sjet.v1i1.188 (date of access: 04.10.2024).

3. Ahmad R. M., Reffat R. M. A comparative study of various daylighting systems in office buildings for improving energy efficiency in Egypt. *Journal of Building Engineering*. 2018. Vol. 18. P. 360–376. DOI:10.1016/j.jobe.2018.04.002 (date of access: 04.12.2024).

4. How Hue works. URL: https://www.philips-hue.com/en-us/explore-hue/how-it-works (Last accessed: 05.10.2024).

5. A Guide to Home Lighting Control. URL: https://assets.lutron.com/a/documents/367-1018_ea.pdf (Last accessed: 05.10.2024).

6. What is the Caseta Smart Hub and when do I need one. URL: https://support.lutron.com/us/en/product/casetawireless/article/product-selection/What-is-the-Caseta-Smart-Hub-and-when-do-I-need-one (Last accessed: 05.10.2024).

7. Lee J., Sim M. K., Hong J. -S. Assessing Decision Tree Stability: A Comprehensive Method for Generating a Stable Decision Tree, *IEEE Access*, 2024 Vol. 12, P. 90061-90072, DOI: 10.1109/ACCESS.2024.3419228.

8. Philips Hue White Paper. URL: https://developers.meethue.com/Philips%20Hue%20-%20Sustainability%20-%20Whitepaper.pdf (Last accessed: 05.10.2024).

9. Lutron: Energy Savings. URL: https://www.lutron.com/asia/Residential-Commercial-Solutions/Pages/Commercial-Solutions/CommercialEnergySavings.aspx (Last accessed: 05.10.2024).

10. Philips Hue: переваги та недоліки системи в розумному будинку. URL: https://comtrade.ua/ua/blog/philips-hue-preimushchestva-i-nedostatki-sistemy-v-umnom-dome/ (дата звернення: 05.10.2024).

11. Dudzik M., Dechnik M., Furtak M. Application of neural networks to lighting systems. *MATEC Web Conf.* 2019 Vol. 282 P. 6 DOI:10.1051/matecconf/201928202069.

12. ESP32-CAM AI-Thinker Pinout Guide: GPIOs Usage Explained. *Random Nerd Tutorials*. URL: https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/ (date of access: 27.11.2024).

13. HC-SR501 PIR Sensor. *Components101*. URL: https://components101.com/sensors/hc-sr501-pir-sensor (date of access: 27.11.2024).

14. RGB LED стрічка WS2812B з адресацією. *Mini-Tech*. URL: https://www.mini-tech.com.ua/ua/rgb-strip-ws2812b-60-led (дата звернення: 27.11.2024).

15. Датчик освітленості цифровий GY302 BH 1750 BH-1750. *Arduino в Україні*. URL: https://arduino.ua/prod1116-datchik-osveshhennosti-cifrovoi-bh1750fvi (дата звернення: 27.11.2024).

16. VK4PK - YP-05 FTDI FT232L USB to TTL UART. *Amateur Radio Station VK4PK - Glenn*. URL: https://www.lyonscomputer.com.au/Electronic-Modules/USB-to-TTL-UART-YP-05-FTDI-FT232L/USB-to-TTL-UART-YP-05-FTDI-FT232L.html (date of access: 27.11.2024).

17. Söderby K., Hylén J. Getting Started with Arduino IDE 2. URL: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/ (date of access: 27.11.2024).

18. Welcome to Flask – Flask Documentation (3.1.x). URL: https://flask.palletsprojects.com/en/stable/ (date of access: 27.11.2024).

19. GitHub - serengil/deepface: A Lightweight Face Recognition and Facial Attribute Analysis (Age, Gender, Emotion and Race) Library for Python. *GitHub*. URL: https://github.com/serengil/deepface (date of access: 28.11.2024).

20. Smart LED lighting starter kits | Philips Hue. *Philips Hue*. URL: https://www.philips-hue.com/en-us/products/smart-light-starter-kits (date of access: 28.11.2024).

21. Lutron smart landscape lighting control solutions. *Lutron*.URL: https://assets.lutron.com/a/documents/3685669_caseta_smart_landscaping_solutions_brochure.pdf (date of access: 28.11.2024).

22. Caseta trade brochure. *Lutron*.URL: https://assets.lutron.com/a/documents/3672499ca_caseta_trade_brochure_sg.pdf (date of access: 28.11.2024).

ДОДАТОК А

Код програми

Додаток А.1. Код пристрою

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <BH1750.h>
#include <Adafruit_NeoPixel.h>

// WiFi credentials
const char* ssid = "TP-LINK_1EE8";
const char* password = "homenet.bagi32";

// Server endpoint
const char* serverURL = "http://192.168.0.108:5000/process_data";

// NeoPixel configuration
#define LED_PIN 12
#define NUM_LEDS 30
Adafruit_NeoPixel strip(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);

// BH1750 object
BH1750 lightSensor;

// Async control flag
bool isRequestInProgress = false;

void setup() {
  Serial.begin(115200);

  // WiFi connection
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected!");

  // BH1750 initialization
  Wire.begin();
  if (!lightSensor.begin()) {
    Serial.println("BH1750 initialization failed!");
    while (1);
  }
```

```
  Serial.println("BH1750 initialized!");

  // NeoPixel setup
  strip.begin();
  strip.show(); // Ensure all LEDs are off
}

void sendData(float lightLevel) {
  if (WiFi.status() == WL_CONNECTED && !isRequestInProgress) {
    isRequestInProgress = true;

    HTTPClient http;
    http.begin(serverURL);
    http.addHeader("Content-Type", "application/json");

    // Create JSON payload
    String payload = "{\"lightLevel\": " + String(lightLevel) + "}";
    int httpResponseCode = http.POST(payload);

    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println("Response JSON: " + response);

      // Parse JSON response
      int brightnessIndex = response.indexOf("\"brightness\":") + 13;
      int colorDataIndex = response.indexOf("\"colorData\": [") + 13;

      if (brightnessIndex > 12 && colorDataIndex > 12) {
        String brightnessStr = response.substring(brightnessIndex, response.indexOf(",",
brightnessIndex));
        String colorDataStr = response.substring(colorDataIndex, response.indexOf("]",
colorDataIndex));

        int brightness = brightnessStr.toInt();
        Serial.println("Brightness: " + String(brightness));
        Serial.println("Color Data: " + colorDataStr);

        // Parse color data
        int r = colorDataStr.substring(0, colorDataStr.indexOf(",")).toInt();
        int g = colorDataStr.substring(colorDataStr.indexOf(",") + 1,
colorDataStr.lastIndexOf(",")).toInt();
        int b = colorDataStr.substring(colorDataStr.lastIndexOf(",") + 1).toInt();

        // Set brightness and color to LED strip
        strip.setBrightness(brightness);
        for (int i = 0; i < NUM_LEDS; i++) {
          strip.setPixelColor(i, strip.Color(r, g, b));
```

```
            }
            strip.show();
          }
        } else {
          Serial.println("Error sending data: " + String(httpResponseCode));
        }

        http.end();
        isRequestInProgress = false;
    } else {
        Serial.println("WiFi not connected or request in progress.");
    }
}

void loop() {
    if (!isRequestInProgress) {
        float lightLevel = lightSensor.readLightLevel();
        Serial.println("Sending light level data...");
        sendData(lightLevel);
        delay(5000); // Delay to avoid frequent requests
    }
}
```

Додаток А.2 Код серверу

```
import cv2
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from deepface import DeepFace
import os
import base64
from concurrent.futures import ThreadPoolExecutor
from datetime import datetime, timedelta
app = Flask(__name__)
# Конфігурація бази даних
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///preferences.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
# Directory to save uploaded photos
UPLOAD_FOLDER = 'uploads'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)
# Модель для збереження preferences
class Preferences(db.Model):
    user_id = db.Column(db.String(50), nullable=False, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    brightness = db.Column(db.Float, nullable=False)
    color_mode = db.Column(db.String(20), nullable=False)
    auto_color_mode = db.Column(db.Boolean, nullable=False)
    warm_light_start_time = db.Column(db.String(10), nullable=False)
    photo_path = db.Column(db.String(200), nullable=False)
    lightingTimeout = db.Column(db.Float, nullable=False)
    adaptiveLevel = db.Column(db.Float, nullable=False)
    priority = db.Column(db.Integer, nullable=False)

class ModeSetting(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    mode = db.Column(db.String(50), nullable=False)
# Ініціалізація бази даних
with app.app_context():
    db.create_all()
room_users = []
def encode_photo(photo_path):
    with open(photo_path, 'rb') as photo_file:
        return base64.b64encode(photo_file.read()).decode('utf-8')
@app.route('/mode', methods=['POST'])
def update_mode():
    data = request.get_json()
    if not data or 'mode' not in data:
        return jsonify({"error": "Invalid data"}), 400
```

```
    # Оновлення або створення запису
    setting = ModeSetting.query.first()
    if setting:
        setting.mode = data['mode']
    else:
        setting = ModeSetting(mode=data['mode'])
        db.session.add(setting)
    db.session.commit()
    return jsonify({"message": "Mode updated successfully"}), 200
@app.route('/mode', methods=['GET'])
def get_mode():
    setting = ModeSetting.query.first()
    if setting:
        return jsonify({"mode": setting.mode}), 200
    return jsonify({"mode": "adaptive"}), 200
@app.route('/test', methods=['POST'])
def test():
    # Отримання даних від ESP32-CAM
    data = request.data.decode('utf-8')
    print("Received data:", data)
    # Формування відповіді для ESP32-CAM
    response = {
        'status': 'success',
        'message': 'Data received successfully',
        'info': 'Server is operational'
    }
    return jsonify(response), 200
@app.route('/preferences', methods=['POST'])
def add_preferences():
    try:
        print(request.form)
        user_id = request.form.get('id', '123')
        name = request.form.get('name', 'undefined')
        brightness = float(request.form.get('brightness', 0))
        color_mode = request.form.get('colorMode', 'cool')
        auto_color_mode = request.form.get('autoColorMode', 'false') == 'true'
        warm_light_start_time = request.form.get('warmLightStartTime', '00:00')
        lightingTimeout = float(request.form.get('lightingTimeout', '5.0'))
        adaptiveLevel = float(request.form.get('adaptiveLevel', 50))
        priority = int(request.form.get('priority', 1))
        # Process the uploaded photo file
        if 'photo' not in request.files:
            return jsonify({'error': 'No photo uploaded'}), 400
        photo = request.files['photo']
        photo_filename = os.path.join(UPLOAD_FOLDER, user_id + '.jpg')
        photo.save(photo_filename)
        new_preference = Preferences(
```

```
            user_id=user_id,
            name=name,
            brightness=brightness,
            color_mode=color_mode,
            auto_color_mode=auto_color_mode,
            warm_light_start_time=warm_light_start_time,
            photo_path=photo_filename,
            lightingTimeout=lightingTimeout,
            adaptiveLevel=adaptiveLevel,
            priority=priority
        )
        db.session.add(new_preference)
        db.session.commit()
        return jsonify({
            'message': 'Preferences and photo saved successfully',
            'data': {
                'id': new_preference.user_id,
                'name': new_preference.name,
                'brightness': new_preference.brightness,
                'colorMode': new_preference.color_mode,
                'autoColorMode': new_preference.auto_color_mode,
                'warmLightStartTime': new_preference.warm_light_start_time,
                'photoPath': new_preference.photo_path,
                'lightingTimeout': new_preference.lightingTimeout,
                'adaptiveLevel': new_preference.adaptiveLevel,
                'priority': new_preference.priority
            }
        }), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
@app.route('/preferences/<user_id>', methods=['POST'])
def update_preferences(user_id):
    try:
        # Перевірка, чи існує користувач
        preference = Preferences.query.filter_by(user_id=user_id).first()
        if not preference:
            return jsonify({'error': 'User not found'}), 404
        # Оновлення полів
        name = request.form.get('name', preference.name)
        brightness = float(request.form.get('brightness', preference.brightness))
        color_mode = request.form.get('colorMode', preference.color_mode)
        auto_color_mode = request.form.get('autoColorMode', str(preference.auto_color_mode).lower())
== 'true'
        warm_light_start_time = request.form.get('warmLightStartTime',
preference.warm_light_start_time)
        lightingTimeout = request.form.get('lightingTimeout', preference.lightingTimeout)
        adaptiveLevel = request.form.get('adaptiveLevel', preference.adaptiveLevel)
```

```
        priority = request.form.get('priority', preference.priority)
        # Оновлення фото, якщо воно надане
        if 'photo' in request.files:
            photo = request.files['photo']
            photo_filename = os.path.join(UPLOAD_FOLDER, user_id + '.jpg')
            photo.save(photo_filename)
            preference.photo_path = photo_filename
        # Застосування змін
        preference.name = name
        preference.brightness = brightness
        preference.color_mode = color_mode
        preference.auto_color_mode = auto_color_mode
        preference.warm_light_start_time = warm_light_start_time
        preference.lightingTimeout = lightingTimeout
        preference.adaptiveLevel = adaptiveLevel
        preference.priority = priority
        db.session.commit()
        global room_users
        for user in room_users:
            if user["id"] == user_id:
                user["brightness"] = brightness
                user["name"] = name
                user["color_mode"] = color_mode
                user["auto_color_mode"] = auto_color_mode
                user["warm_light_start_time"] = warm_light_start_time
                user["lightingTimeout"] = lightingTimeout
                user["adaptiveLevel"] = adaptiveLevel
                user["priority"] = priority
        return jsonify({
            'message': 'Preferences updated successfully',
            'data': {
                'id': preference.user_id,
                'name': preference.name,
                'brightness': preference.brightness,
                'colorMode': preference.color_mode,
                'autoColorMode': preference.auto_color_mode,
                'warmLightStartTime': preference.warm_light_start_time,
                'photoPath': preference.photo_path,
                'lightingTimeout': preference.lightingTimeout,
                'adaptiveLevel': preference.adaptiveLevel,
                'priority': preference.priority
            }
        }), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
# DELETE маршрут
@app.route('/preferences/<user_id>', methods=['DELETE'])
```

```
def delete_preference(user_id):
    try:
        # Пошук користувача в базі даних за user_id
        preference = Preferences.query.filter_by(user_id=user_id).first()
        if not preference:
            return jsonify({'error': 'Preference not found'}), 404

        os.remove(os.path.join(UPLOAD_FOLDER, user_id + '.jpg'))
        # Видалення запису
        db.session.delete(preference)
        db.session.commit()
        return jsonify({'message': f'Preference with id {user_id} deleted successfully'}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
@app.route('/preferences', methods=['GET'])
def get_all_preferences():
    try:
        # Fetch all preferences from the database
        preferences = Preferences.query.all()
        # Convert each record into a dictionary
        preferences_list = [
            {
                'id': preference.user_id,
                'name': preference.name,
                'brightness': preference.brightness,
                'colorMode': preference.color_mode,
                'autoColorMode': preference.auto_color_mode,
                'warmLightStartTime': preference.warm_light_start_time,
                'photo': encode_photo(preference.photo_path),
                'lightingTimeout': preference.lightingTimeout,
                'adaptiveLevel': preference.adaptiveLevel,
                'priority': preference.priority,
            }
            for preference in preferences
        ]
        # Return all preferences as a JSON response
        return jsonify(preferences_list), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
@app.route('/update_priorities', methods=['POST'])
def update_priorities():
    try:
        # Отримання даних із запиту
        data = request.get_json()
        if not data or 'preferences' not in data:
            return jsonify({'error': 'Invalid data format'}), 400
        preferences = data['preferences']
```

```
        # Оновлення пріоритетів у базі даних
        for preference in preferences:
            user_id = preference.get('id')
            new_priority = preference.get('priority')
            if user_id is None or new_priority is None:
                return jsonify({'error': 'Missing user id or priority'}), 400
            # Пошук користувача в базі даних
            user = Preferences.query.filter_by(user_id=user_id).first()
            if not user:
                return jsonify({'error': f'User with id {user_id} not found'}), 404
            # Оновлення пріоритету в базі даних
            user.priority = new_priority
            # Оновлення пріоритету в кімнаті
            global room_users
            for room_user in room_users:
                if room_user["id"] == user_id:
                    room_user["priority"] = new_priority
                    break
        # Підтвердження змін у базі
        db.session.commit()
        return jsonify({'message': 'Priorities updated successfully'}), 200
    except Exception as e:
        print(f"Error updating priorities: {e}")
        return jsonify({'error': str(e)}), 500
@app.route('/process_data', methods=['POST'])
def process_data():
    try:
        # Отримання даних із запиту
        data = request.get_json()
        light_level = data.get("lightLevel", None)  # Рівень природного освітлення (в lx)
        photo_base64 = data.get("photo", None)
        if light_level is None or photo_base64 is None:
            return jsonify({'error': 'Missing lightLevel or photo'}), 400
        # Декодування фото
        temp_photo_path = os.path.join('temp_photo', 'temp_photo.jpg')
        with open(temp_photo_path, "wb") as f:
            f.write(base64.b64decode(photo_base64))
        # Пошук користувачів на фото
        try:
            df = DeepFace.find(temp_photo_path, UPLOAD_FOLDER)
        except Exception as e:
            print(f"Error detecting faces: {e}")
            return jsonify({'error': 'Error detecting faces'}), 500
        if len(df) == 0:
            return jsonify({'error': 'No recognized users found in the image'}), 404
        # Ідентифікація користувачів
        recognized_users = []
```

```
for i in range(len(df)):
    user_id = df[i].values[0][0].split('/')[1].split('.')[0]
    user = Preferences.query.filter_by(user_id=user_id).first()
    print(f'Detected {user.name}')
    if user and user not in recognized_users:
        recognized_users.append(user)
# Оновлення списку користувачів у кімнаті
global room_users
now = datetime.now()
room_users = [
    user for user in room_users
    if (now - user["added_at"]).total_seconds() / 60 < float(user["lightingTimeout"])
]
for user in recognized_users:
    existing_user = next((u for u in room_users if u["id"] == user.user_id), None)
    if existing_user:
        existing_user["added_at"] = now
    else:
        room_users.append({
            "id": user.user_id,
            "name": user.name,
            "brightness": user.brightness,
            "color_mode": user.color_mode,
            "auto_color_mode": user.auto_color_mode,
            "warm_light_start_time": user.warm_light_start_time,
            "lightingTimeout": user.lightingTimeout,
            "adaptiveLevel": user.adaptiveLevel,
            "priority": user.priority,
            "added_at": now
        })
# Якщо список користувачів порожній, вимикаємо світло
if not room_users:
    return jsonify({"brightness": 0, "colorData": [0, 0, 0]}), 200
# Отримання поточного режиму роботи
mode_setting = ModeSetting.query.first()
current_mode = mode_setting.mode if mode_setting else "adaptive"
# Максимальний рівень освітлення LED-стрічки у люксах
max_led_lux = 1000  # Замініть на фактичне значення
# Дерево рішень для визначення параметрів освітлення
if current_mode == "priority":
    print('Priority mode')
   highest_priority_user = min(room_users, key=lambda user: user["priority"])
    print(f'Settings of {highest_priority_user["name"]} is used')
    adaptive_factor = float(highest_priority_user["adaptiveLevel"]) / 100
    user_brightness = highest_priority_user["brightness"]
    adjusted_brightness = user_brightness * adaptive_factor
    color_mode = highest_priority_user["color_mode"]
```

```
        elif current_mode == "adaptive":
            print('Adaptive mode')
            names = ", ".join([user["name"] for user in room_users])
            print(f'Settings of { names } are used')
            total_weights = sum(float(user["adaptiveLevel"]) / 100 for user in room_users)
            if total_weights > 0:
                weighted_brightness = sum(
                    user["brightness"] * (float(user["adaptiveLevel"]) / 100) for user in room_users
                )
                avg_brightness = weighted_brightness / total_weights
            else:
                avg_brightness = 0
            warm_votes = sum(float(user["adaptiveLevel"]) / 100 for user in room_users if
user["color_mode"] == "warm")
            cool_votes = sum(float(user["adaptiveLevel"]) / 100 for user in room_users if
user["color_mode"] == "cool")
            color_mode = "warm" if warm_votes > cool_votes else "cool"
            avg_adaptive_level = total_weights / len(room_users) if len(room_users) > 0 else 0
            energy_saving_factor = 1 - avg_adaptive_level
            adjusted_brightness = avg_brightness * (1 - energy_saving_factor)
        else:
            adjusted_brightness = 0
            color_mode = "off"
        # Перетворення adjusted_brightness у люкси
        adjusted_lux = adjusted_brightness * max_led_lux / 100
        # Порівняння з природним освітленням
        if light_level >= adjusted_lux:
            adjusted_brightness = 0
            color_mode = "off"
        # Конвертація яскравості в реальне значення
        real_brightness = round(adjusted_brightness * 255 / 100)
        # Перетворення кольору на RGB
        if color_mode == "warm":
            color_data = [255, 180, 100]  # Теплий білий
        elif color_mode == "cool":
            color_data = [180, 200, 255]  # Холодний білий
        else:
            color_data = [0, 0, 0]  # Світло вимкнено
        # Підготовка даних для ESP32-CAM
        response_data = {
            "brightness": real_brightness,
            "colorData": color_data
        }
        print(f"Sending to ESP32-CAM: {response_data}")
        return jsonify(response_data), 200
    except Exception as e:
        print(f"Error processing data: {e}")
```

```
        return jsonify({'error': str(e)}), 500
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Додаток А.3 Код застосунку Flutter (main.dart)

```
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:masters_project/screens/home_screen.dart';
import 'package:masters_project/screens/onboarding_screen/onboarding_provider.dart';
import 'package:masters_project/screens/onboarding_screen/onboarding_screen.dart';
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const ProviderScope(child: App()));
}
class App extends ConsumerWidget {
  const App({super.key});
  @override
  Widget build(BuildContext context, ref) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Masters Project',
      theme: ThemeData(
        // colorScheme: ColorScheme.fromSeed(seedColor: Colors.orange[400]!),
        useMaterial3: true,
      ),
      home: ref.watch(onboardingNotifierProvider)
          ? const HomeScreen()
          : const OnboardingScreen(),
    );
  }
}
```

Додаток А.4 Код застосунку Flutter (home_screen.dart)

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:masters_project/providers/data_processing_mode.dart';
import 'package:masters_project/providers/lighting_preference_provider.dart';
import 'package:masters_project/screens/preference_screen.dart';
import 'package:path_provider/path_provider.dart';
class HomeScreen extends ConsumerStatefulWidget {
  const HomeScreen({
    super.key,
  });
  @override
  ConsumerState<HomeScreen> createState() => _HomeScreenState();
}
class _HomeScreenState extends ConsumerState<HomeScreen> {
  bool _isLoading = true;
  Future<File> getImageFileFromAssets(String path) async {
    final byteData = await rootBundle.load('assets/$path');
    final file = File('${(await getTemporaryDirectory()).path}/$path');
    await file.create(recursive: true);
    await file.writeAsBytes(byteData.buffer
        .asUint8List(byteData.offsetInBytes, byteData.lengthInBytes));
    return file;
  }
  @override
  void initState() {
    super.initState();
    Future.delayed(const Duration(seconds: 2), () {
      setState(() {
        _isLoading = false;
      });
    });
  }
  @override
  Widget build(BuildContext context) {
    final preferences = ref.watch(lightingPreferencesProvider);
    final processingMode = ref.watch(dataProcessingModeProvider);
    return Scaffold(
        appBar: AppBar(
          title: const Text(
            'Smart Light Control',
          ),
        ),
        body: _isLoading
```

```
? const Center(
    child: CircularProgressIndicator(),
  )
: Padding(
    padding:
        const EdgeInsets.symmetric(horizontal: 15, vertical: 10),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 8.0),
          child: Row(
            mainAxisSize: MainAxisSize.min,
            mainAxisAlignment: MainAxisAlignment.start,
            children: [
              DropdownButton<String>(
                value: processingMode.mode,
                items: const [
                  DropdownMenuItem(
                      value: "adaptive", child: Text("Адаптивний")),
                  DropdownMenuItem(
                      value: "priority",
                      child: Text("Приорітетний")),
                ],
                onChanged: (newValue) {
                  ref
                      .read(dataProcessingModeProvider.notifier)
                      .updateMode(newValue ?? 'adaptive');
                },
              ),
              const SizedBox(
                width: 10,
              ),
              const Text('режим')
            ],
          ),
        ),
        Expanded(
          child: ReorderableListView(
            children: [
              for (final preference in preferences)
                Column(
                  key: ValueKey(preference.id),
                  children: [
                    const Divider(),
                    ListTile(
                      leading: CircleAvatar(
```

```
  radius: 26,
  backgroundImage:
      FileImage(preference.photo),
),
title: Text(preference.name),
subtitle: Text(
   'Рівень світла: ${preference.brightness.toStringAsFixed(0)} %'),
trailing: Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    InkWell(
       onTap: () {
         Navigator.of(context).push(MaterialPageRoute(
            builder: (context) =>
               LightingPreferencesPage(
                  id: preference.id,
                  name: preference.name,
                  selectedImage:
                     preference.photo,
                  brightness: preference
                     .brightness,
                  colorMode: preference
                     .colorMode,
                  autoColorMode:
                     preference
                        .autoColorMode,
                  warmLightStartTime:
                     preference
                        .warmLightStartTime,
                  lightingTimeout:
                     preference
                        .lightingTimeout,
                  priority: preference
                     .priority)));
       },
       child: const Icon(Icons.edit)),
    const SizedBox(
      width: 10,
    ),
    InkWell(
       onTap: () {
         showDialog(
            context: context,
            builder: (ctx) => AlertDialog(
                 title: const Text(
                    'Видалення...'),
                 content: Text(
```

```
                    "Бажаєте видалити налаштування ${preference.name}?"),
                actions: [
                  TextButton(
                    onPressed: () {
                      Navigator.of(
                          context)
                        .pop();
                    },
                    child: const Text(
                      'Hi')),
                  TextButton(
                    onPressed: () {
                      ref
                        .read(lightingPreferencesProvider
                          .notifier)
                        .deletePreference(
                          preference
                            .id);
                      Navigator.of(
                          context)
                        .pop();
                    },
                    child: const Text(
                      'Так'))
                ],
              ));
        },
        child: const Icon(Icons.delete)),
    const SizedBox(
      width: 10,
    ),
    if (processingMode.mode == 'priority')
      ReorderableDragStartListener(
        index:
          preferences.indexOf(preference),
        child: InkWell(
          onTap: () {},
          child: const Icon(
            Icons.drag_handle)),
      ),
  ],
 ),
),
if (preferences.indexOf(preference) ==
  preferences.length - 1)
 const Divider()
],
```

```
                    ),
                  ],
                  onReorder: (oldIndex, newIndex) {
                    ref
                        .read(lightingPreferencesProvider.notifier)
                        .reorder(oldIndex, newIndex);
                  },
                ),
              ),
              if (preferences.isNotEmpty)
                const SizedBox(
                  height: 10,
                ),
              SizedBox(
                height: 50,
                width: double.infinity,
                child: OutlinedButton.icon(
                    onPressed: () async {
                      // File file =
                      //     await getImageFileFromAssets('ryan.png');
                      Navigator.of(context).push(MaterialPageRoute(
                          builder: (context) => LightingPreferencesPage(
                              // selectedImage: file,
                              )));
                    },
                    style: OutlinedButton.styleFrom(
                        shape: RoundedRectangleBorder(
                          borderRadius: BorderRadius.circular(18.0),
                        ),
                        side: BorderSide(
                            width: 2,
                            color:
                                Theme.of(context).colorScheme.primary)),
                    label: const Text(
                      'Додати нового користувача',
                    ),
                    icon: const Icon(
                      Icons.add,
                    )),
              ),
            ],
          ),
        ));
  }
}
```

Додаток А.5 Код застосунку Flutter (preference_screen.dart)

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:masters_project/providers/lighting_preference_provider.dart';
import 'package:masters_project/widgets/image_inputs.dart';
class LightingPreferencesPage extends ConsumerStatefulWidget {
  const LightingPreferencesPage(
      {super.key,
      this.id,
      this.brightness = 75,
      this.colorMode = 'cool',
      this.autoColorMode = false,
      this.warmLightStartTime = const TimeOfDay(hour: 18, minute: 0),
      this.lightingTimeout = 5,
      this.adaptiveLevel = 50,
      this.priority,
      this.selectedImage,
      this.name});
  final String? id;
  final double brightness;
  final String colorMode;
  final bool autoColorMode;
  final TimeOfDay warmLightStartTime;
  final double lightingTimeout;
  final File? selectedImage;
  final double adaptiveLevel;
  final String? name;
  final int? priority;
  @override
  ConsumerState<LightingPreferencesPage> createState() =>
      _LightingPreferencesCopyPageState();
}
class _LightingPreferencesCopyPageState
    extends ConsumerState<LightingPreferencesPage> {
  TextEditingController? textEditingController;
  late double _brightness;
  late String _colorMode;
  late bool _autoColorMode;
  late TimeOfDay _warmLightStartTime;
  late double _lightingTimeout;
  late double _adaptiveLevel;
  late File? _selectedImage;
  @override
  void initState() {
    textEditingController = TextEditingController();
```

```
  textEditingController!.text = widget.name ?? '';
  _brightness = widget.brightness;
  _colorMode = widget.colorMode;
  _autoColorMode = widget.autoColorMode;
  _warmLightStartTime = widget.warmLightStartTime;
  _lightingTimeout = widget.lightingTimeout;
  _selectedImage = widget.selectedImage;
  _adaptiveLevel = widget.adaptiveLevel;
  super.initState();
}
@override
void dispose() {
  textEditingController!.dispose();
  super.dispose();
}
// Метод для перетворення TimeOfDay у String
String formatTimeOfDay(TimeOfDay time) {
  return '${time.hour.toString().padLeft(2, '0')}:${time.minute.toString().padLeft(2, '0')}';
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text("Налаштування освітлення"),
    ),
    body: Column(
      children: [
        Expanded(
          child: Padding(
            padding: const EdgeInsets.all(8.0),
            child: ListView(
              // physics: const NeverScrollableScrollPhysics(),
              children: [
                ImageInput(
                  image: _selectedImage,
                  onPickImage: (image) {
                    setState(() {
                      _selectedImage = image;
                    });
                  },
                ),
                const SizedBox(
                  height: 15,
                ),
                Padding(
                  padding: const EdgeInsets.symmetric(horizontal: 16.0),
                  child: TextField(
```

```
    controller: textEditingController,
    decoration: const InputDecoration(
      border: OutlineInputBorder(),
      labelText: 'Enter your username',
    ),
    onSubmitted: (value) {
      setState(() {});
    },
  ),
),
// Яскравість
ListTile(
  title: const Text("Яскравість"),
  subtitle: Slider(
    value: _brightness,
    min: 0,
    max: 100,
    divisions: 20,
    label: "${_brightness.round()}%",
    onChanged: (value) {
      setState(() {
        _brightness = value;
      });
    },
  ),
),
// Режим кольору
ListTile(
  title: const Text("Режим кольору"),
  trailing: DropdownButton<String>(
    value: _colorMode,
    items: const [
      DropdownMenuItem(
          value: "cool", child: Text("Холодний білий")),
      DropdownMenuItem(
          value: "warm", child: Text("Теплий білий")),
    ],
    onChanged: (newValue) {
      setState(() {
        if (newValue == "warm") {
          _autoColorMode = false;
        }
        _colorMode = newValue!;
      });
    },
  ),
),
```

```
// Автоматичний перехід на теплий білий
if (_colorMode == "cool")
  CheckboxListTile(
    title: const Text("Автоматичний перехід на теплий білий"),
    value: _autoColorMode,
    onChanged: (newValue) {
      setState(() {
        _autoColorMode = newValue!;
      });
    },
  ),
// Час для теплого білого
if (_autoColorMode)
  ListTile(
    title: const Text("Час початку теплого світла"),
    trailing: TextButton(
      onPressed: () async {
        TimeOfDay? pickedTime = await showTimePicker(
          context: context,
          initialTime: _warmLightStartTime,
        );
        if (pickedTime != null) {
          setState(() {
            _warmLightStartTime = pickedTime;
          });
        }
      },
      child: Text(formatTimeOfDay(_warmLightStartTime)),
    ),
  ),
// Таймаут освітлення
ListTile(
  title: const Text("Таймаут освітлення"),
  subtitle: Slider(
    value: _lightingTimeout,
    min: 1,
    max: 10,
    divisions: 9,
    label: "${_lightingTimeout.round()} хв",
    onChanged: (value) {
      setState(() {
        _lightingTimeout = value;
      });
    },
  ),
),
ListTile(
```

```
            title: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                const Text("Рівень адаптивності"),
                IconButton(
                  icon: const Icon(Icons.info_outline,
                      color: Colors.grey),
                  onPressed: () {
                    showDialog(
                      context: context,
                      builder: (BuildContext context) {
                        return AlertDialog(
                          title: const Text(
                              "Що таке рівень адаптивності?"),
                          content: const Text(
                              "Рівень адаптивності визначає, наскільки система автоматично "
                              "адаптує освітлення відповідно до рівня зовнішнього світла "
                              "та інших умов. Чим вищий рівень, тим більшу роль відіграють
налаштування користувача."),
                          actions: [
                            TextButton(
                              onPressed: () {
                                Navigator.of(context).pop();
                              },
                              child: const Text("Зрозуміло"),
                            ),
                          ],
                        );
                      },
                    );
                  },
                ),
              ],
            ),
            subtitle: Slider(
              value: _adaptiveLevel,
              min: 0,
              max: 100,
              divisions: 10,
              label: "${_adaptiveLevel.round()}%",
              onChanged: (value) {
                setState(() {
                  _adaptiveLevel = value;
                });
              },
            ),
          ),
```

```
      ],
    ),
  ),
),
SizedBox(
  width: double.infinity,
  height: 50,
  child: ElevatedButton(
    style: ElevatedButton.styleFrom(
        shape: const RoundedRectangleBorder(),
        backgroundColor: _selectedImage == null ||
                textEditingController!.text.trim().isEmpty
            ? Colors.grey[400]
            : Theme.of(context).colorScheme.primary),
    onPressed: () {
      if (_selectedImage == null ||
          textEditingController!.text.trim().isEmpty) {
      } else {
        if (widget.id == null) {
          ref
              .read(lightingPreferencesProvider.notifier)
              .addPreference(
                  photo: _selectedImage!,
                  name: textEditingController!.text,
                  brightness: _brightness,
                  colorMode: _colorMode,
                  autoColorMode: _autoColorMode,
                  warmLightStartTime: _warmLightStartTime,
                  lightingTimeout: _lightingTimeout,
                  adaptiveLevel: _adaptiveLevel);
        } else {
          ref
              .read(lightingPreferencesProvider.notifier)
              .updatePreference(
                  id: widget.id!,
                  photo: _selectedImage!,
                  name: textEditingController!.text,
                  brightness: _brightness,
                  colorMode: _colorMode,
                  autoColorMode: _autoColorMode,
                  warmLightStartTime: _warmLightStartTime,
                  lightingTimeout: _lightingTimeout,
                  adaptiveLevel: _adaptiveLevel,
                  priority: widget.priority!);
        }
        Navigator.of(context).pop();
      }
```

```
            },
            child: const Text(
              "Зберегти налаштування",
              style: TextStyle(color: Colors.white),
            ),
          ),
        ),
      ],
    ),
  );
 }
}
```

Додаток А.6 Код застосунку Flutter (lighting_preference_provider.dart)

```
import 'dart:convert';
import 'dart:io';
import 'dart:typed_data';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:http/http.dart' as http;
import 'package:masters_project/models/user_settings.dart';
import 'package:path_provider/path_provider.dart';
class LightingPreferencesNotifier
    extends StateNotifier<List<LightingPreferences>> {
  LightingPreferencesNotifier() : super(const []) {
    loadPreferences();
  }
  String baseUrl = 'http://192.168.0.108:5000/preferences';
  Future<void> loadPreferences() async {
    final response = await http.get(Uri.parse(baseUrl));
    List<LightingPreferences> preferences = await Future.wait(
      (jsonDecode(response.body) as List<dynamic>)
          .map<Future<LightingPreferences>>((value) async {
        final photoData = base64.decode(
            value['photo']); // This is Uint8List or Base64-encoded string
        value['photo'] =
            await convertUint8ListToFile(photoData, '${value['id']}.jpg');
        return LightingPreferences.fromJson(
            value); // Return LightingPreferences instance
      }),
    );
    state = [...preferences];
  }
  void reorder(int oldIndex, int newIndex) {
    final updatedList = List<LightingPreferences>.from(state);
    if (oldIndex < newIndex) {
      newIndex -= 1;
    }
    final movedItem = updatedList.removeAt(oldIndex);
    updatedList.insert(newIndex, movedItem);
    // Update priorities
    final prioritizedList = updatedList
        .asMap()
        .entries
        .map((entry) => entry.value.copyWith(priority: entry.key + 1))
        .toList();
    state = prioritizedList;
    _sendUpdatedPrioritiesToServer(prioritizedList);
  }
```

```
Future<void> _sendUpdatedPrioritiesToServer(
    List<LightingPreferences> preferences) async {
  final url = Uri.parse(
      'http://192.168.0.108:5000/update_priorities'); // Замініть на ваш URL
  final data = preferences
      .map((preference) => {
            'id': preference.id,
            'priority': preference.priority,
          })
      .toList();
  final response = await http.post(
    url,
    headers: {'Content-Type': 'application/json'},
    body: jsonEncode({'preferences': data}),
  );
  if (response.statusCode == 200) {
    print("Priorities updated successfully");
  } else {
    print("Failed to update priorities: ${response.statusCode}");
  }
}
Future<File> convertUint8ListToFile(Uint8List data, String fileName) async {
  // Get the temporary directory of the app
  final tempDir = await getTemporaryDirectory();
  // Create the file path
  final filePath = '${tempDir.path}/$fileName';
  // Write the Uint8List data to the file
  final file = File(filePath);
  return await file.writeAsBytes(data);
}
void addPreference(
    {required File photo,
    required String name,
    required double brightness,
    required String colorMode,
    required bool autoColorMode,
    required TimeOfDay warmLightStartTime,
    required double lightingTimeout,
    required double adaptiveLevel}) async {
  final priority = (state.length + 1);
  LightingPreferences newPreferences = LightingPreferences(
      name: name,
      brightness: brightness,
      colorMode: colorMode,
      autoColorMode: autoColorMode,
      photo: photo,
      warmLightStartTime: warmLightStartTime,
```

```
      lightingTimeout: lightingTimeout,
      adaptiveLevel: adaptiveLevel,
      priority: priority);
  print(newPreferences);
  await sendPreferencesWithPhoto(newPreferences, baseUrl);
  state = [newPreferences, ...state];
 }
 Future<void> sendPreferencesWithPhoto(
    LightingPreferences preferences, String endpont) async {
  final url = Uri.parse(endpont);
  var request = http.MultipartRequest('POST', url);
  // Add JSON fields as form fields
  preferences.toJson().forEach((key, value) {
    if (key != 'photoPath') {
      // Exclude photo path as it's being sent as a file
      request.fields[key] = value.toString();
    }
  });
  // Add the photo file
  request.files.add(await http.MultipartFile.fromPath(
    'photo',
    preferences.photo.path,
  ));
  try {
    var response = await request.send();
    if (response.statusCode == 200) {
      var responseData = await response.stream.bytesToString();
      print('Preferences and photo sent successfully: $responseData');
    } else {
      print('Failed to send preferences: ${response.statusCode}');
    }
  } catch (e) {
    print('Error: $e');
  }
 }
 void updatePreference(
    {required String id,
    required File photo,
    required String name,
    required double brightness,
    required String colorMode,
    required bool autoColorMode,
    required TimeOfDay warmLightStartTime,
    required double lightingTimeout,
    required double adaptiveLevel,
    required int priority}) async {
  LightingPreferences updatedPreferences = LightingPreferences(
```

```
      id: id,
      name: name,
      brightness: brightness,
      colorMode: colorMode,
      autoColorMode: autoColorMode,
      photo: photo,
      warmLightStartTime: warmLightStartTime,
      lightingTimeout: lightingTimeout,
      adaptiveLevel: adaptiveLevel,
      priority: priority);
   state = state.where((preference) => preference.id != id).toList();
   state = [updatedPreferences, ...state];
   sendPreferencesWithPhoto(updatedPreferences, '$baseUrl/$id');
  }
  Future<void> deletePreference(String id) async {
   final url = Uri.parse('$baseUrl/$id');
   try {
    final response = await http.delete(url);
    if (response.statusCode == 200) {
     print('Preferences deleted successfully: ${response.body}');
    } else {
     print('Failed to delete preferences: ${response.statusCode}');
    }
   } catch (e) {
    print('Error: $e');
   }
   state = state.where((preference) => preference.id != id).toList();
  }
}
final lightingPreferencesProvider = StateNotifierProvider<
    LightingPreferencesNotifier, List<LightingPreferences>>((ref) {
  return LightingPreferencesNotifier();
});
```

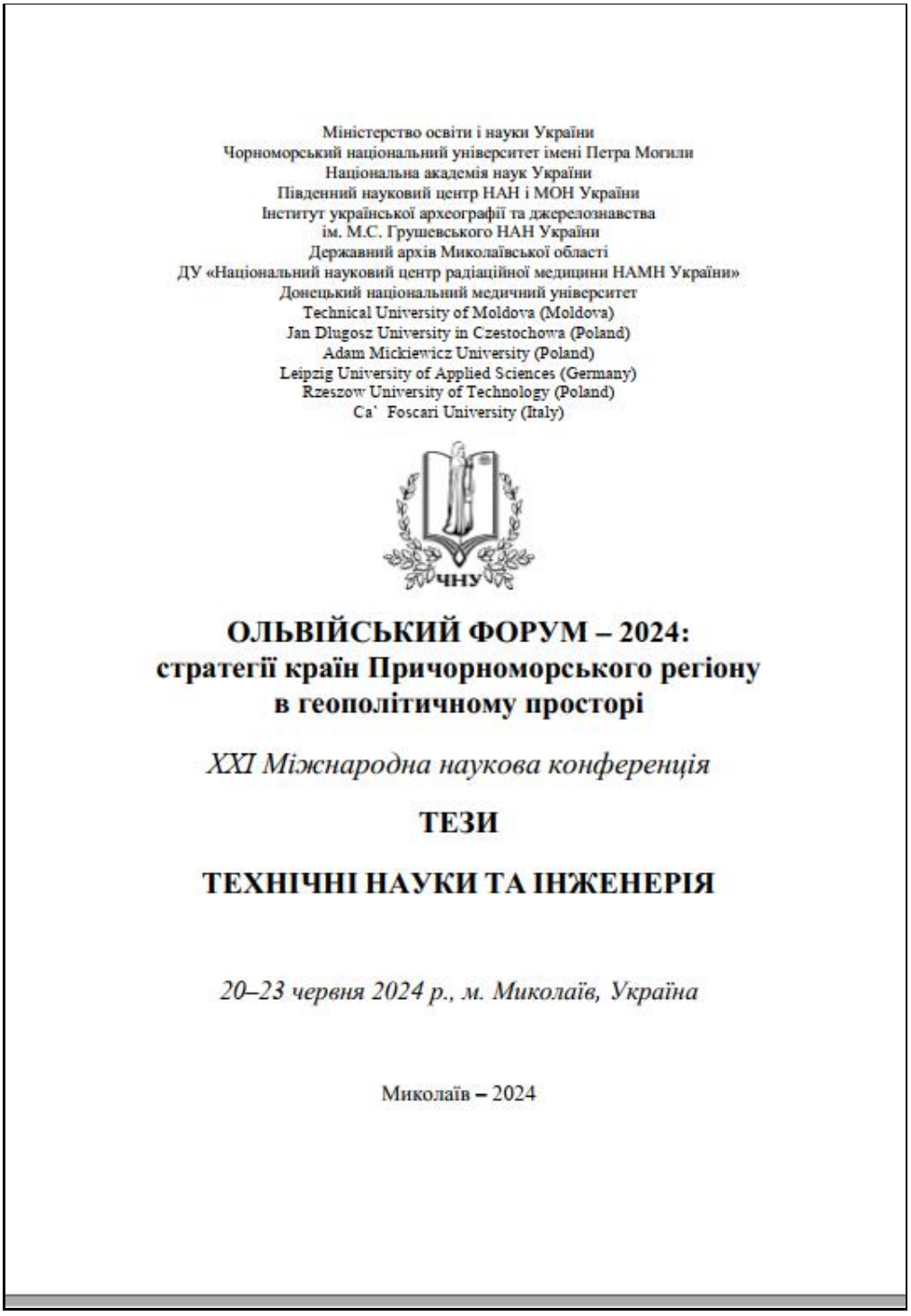
Додаток А.7 Код застосунку Flutter (data_processing_mode.dart)

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:http/http.dart' as http;

import 'package:masters_project/models/data_processing_mode_model.dart';

class DataProcessingModeNotifier extends StateNotifier<DataProcessingMode> {
 DataProcessingModeNotifier() : super(DataProcessingMode('')) {
  fetchModeFromServer();
 }

 String baseUrl = 'http://192.168.0.108:5000/mode';

 void updateMode(String newMode) {
  state = DataProcessingMode(newMode);
  sendSettingsToServer();
 }

 Future<void> sendSettingsToServer() async {
  final url = Uri.parse(baseUrl);
  final response = await http.post(
   url,
   headers: {'Content-Type': 'application/json'},
   body: state.toJson(),
  );

  if (response.statusCode == 200) {
   print('Settings updated successfully on the server');
  } else {
   print('Failed to update settings on the server');
  }
 }

 Future<void> fetchModeFromServer() async {
  try {
   final response = await http.get(Uri.parse(baseUrl));

   if (response.statusCode == 200) {
    state = DataProcessingMode.fromJson(
      response.body); // Повертає режим або "adaptive" за замовчуванням
   } else {
    throw Exception('Failed to load mode');
   }
  } catch (e) {
   print('Error fetching mode: $e');
  }
```

```
  }
}

final dataProcessingModeProvider =
    StateNotifierProvider<DataProcessingModeNotifier, DataProcessingMode>(
        (ref) {
  return DataProcessingModeNotifier();
});
```

ДОДАТОК Б

Апробація кваліфікаційної роботи



Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
Національна академія наук України
Південний науковий центр НАН і МОН України
Інститут української археографії та джерелознавства
ім. М.С. Грушевського НАН України
Державний архів Миколаївської області
ДУ «Національний науковий центр радіаційної медицини НАМН України»
Донецький національний медичний університет
Technical University of Moldova (Moldova)
Jan Dlugosz University in Czestochowa (Poland)
Adam Mickiewicz University (Poland)
Leipzig University of Applied Sciences (Germany)
Rzeszow University of Technology (Poland)
Ca' Foscari University (Italy)

ЧНУ

**ОЛЬВІЙСЬКИЙ ФОРУМ – 2024:
стратегії країн Причорноморського регіону
в геополітичному просторі**

XXI Міжнародна наукова конференція

ТЕЗИ

ТЕХНІЧНІ НАУКИ ТА ІНЖЕНЕРІЯ

20–23 червня 2024 р., м. Миколаїв, Україна

Миколаїв – 2024

ПІДСЕКЦІЯ: Комп'ютерна інженерія

Перелік джерел посилання

1. Відеоспостереження на виробництві. URL: https://dneprsecurity.com/v-promishlennosti/videonabljudenie-na-proizvodstve.html (дата звернення: 30.04.2024).
2. Мінікомп'ютер Raspberry PI 4 Model B 4 GB (RPI4-MODBP-4GB). URL: https://rozetka.com.ua/ua/raspberry-pi-rpi4-modbp-4gb/p316784920/ (дата звернення: 29.04.2024).
3. GPIO: все про підключення Raspberry Pi 4 і 3. URL: https://www.hwlibre.com/uk/gpio-%D0%BC%D0%B0%D0%BB%D0%B8%D0%BD%D0%B0-%D0%BF%D1%96/ (дата звернення: 15.04.2024).

УДК 004.85:004.031.6

Козявко Л. О.,
магістрант кафедри комп'ютерної інженерії,
Пузирьов С. В.,
канд. фіз.-мат. наук, доцент кафедри комп'ютерної інженерії,
ЧНУ імені Петра Могили, м. Миколаїв, Україна

АДАПТИВНЕ УПРАВЛІННЯ ОСВІТЛЕННЯМ В ПРИМІЩЕННІ ЗА ДОПОМОГОЮ ДАТЧИКІВ ТА АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ

У сучасному світі, де проблема енергозбереження стає все актуальнішою через зростаючі тарифи на енергію та потребу захисту навколишнього середовища, створення ефективних систем управління енергоспоживанням у будівлях набуває особливої важливості. Одним із ключових аспектів цього є оптимізація використання освітлення, яке є значним споживачем електрики в комерційних та житлових приміщеннях. Адаптивне керування освітленням може значно скоротити енергоспоживання, автоматично регулюючи інтенсивність та тривалість включення світлових приладів в залежності від потреб користувачів та умов навколишнього середовища [1].

Автоматизація управління освітленням з використанням датчиків та алгоритмів машинного навчання дозволяє не тільки знизити витрати на енергію, але й підвищити комфорт користування приміщеннями. Використання сучасних технологій, таких як датчики освітленості, присутності людей, та часу доби, у поєднанні з алгоритмами машинного навчання, які аналізують та прогнозують поведінку користувачів та