Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Система логування та аналізу умов зйомки для плівкової фотографії

(тема)

Виконав:
здобувач 3 року навчання,
групи КІУКІу-22-1
Даніїл РАПТАНОВ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: доц. Георгій ІВАЩЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Раптанову Даніілу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система логування та аналізу умов зйомки для плівкової фотографії

затверджена наказом по університету від " 26 " травня 2025 р. № 425 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 14 липня 2025 р.

3. Вхідні дані до роботи 1) апаратне забезпечення: платформи Arduino та Лілка, набір сенсорів; 2) документація платформи Node.js; 3) СКБД PostgreSQL; 4) середовище розробки PlatformIO.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз проблеми та огляд існуючих рішень;
2) вибір технології розробки та інструментальних засобів;
3) розробка алгоритмічного забезпечення;
4) реалізація апаратних та програмних модулів;
5) відлагодження програмних модулів;
6) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	10.06.25-13.06.25	
2	Вибір технології розробки та	14.06.25-17.06.25	
	інструментальних засобів		
3	Розробка алгоритмічного забезпечення	18.06.25-21.06.25	
4	Розробка програмних модулів	23.06.25-28.06.25	
5	Відлагодження програмних модулів	30.06.25-02.07.25	
6	Оформлення матеріалів кваліфікаційної роботи	03.07.25-05.07.25	
7	Подання кваліфікаційної роботи керівникові та	07.07.25-09.07.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	10.07.25-11.07.25	
	вання		

Дата видачі завдання " 09 " червня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Георгій ІВАЩЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 126 с., 36 рис., 2 дод., 34 джерел.

МІКРОКОНТРОЛЕРИ, КЛІЄНТ-СЕРВЕРНИЙ ЗАСТОСУНОК, ЛОГУВАННЯ, АНАЛІЗ, ФОТОГРАФІЯ, СИНХРОНІЗАЦІЯ, ВІДДАЛЕНЕ КЕРУВАННЯ, WI-FI.

Метою кваліфікаційної роботи є створення системи логування та аналізу умов зйомки для плівкової фотографії.

У ході виконання кваліфікаційної роботи запропоновано підхід, який включає реєстрацію світлових даних на мікроконтролерному рівні та їхню централізовану обробку, зберігання та інтерпретацію у вебсередовищі. Усі модулі проекту (апаратна частина, програмна логіка та інтерфейси користувача) відкриті до подальшого розширення.

У якості основного технологічного стеку обрана апаратна платформа Lilka на базі ESP32, що забезпечує надійний збір та попередню обробку даних. Бекенд був реалізований на Node.js у поєднанні з PostgreSQL. Це надає можливість масштабування й структурованого зберігання даних. Концепція модульності та відкритості дозволяє поступово інтегрувати нові сенсори, аналітичні алгоритми та механізми обробки подій без порушення сумісності.

ABSTRACT

Bachelor’s thesis: 126 pages, 36 figures, 2 appendices, 34 sources.

MICROCONTROLLERS, CLIENT-SERVER APPLICATION, LOGGING, ANALYSIS, PHOTOGRAPHY, SYNCHRONIZATION, REMOTE CONTROL, WI-FI.

The purpose of the qualification work is to create a system for logging and analyzing shooting conditions for film photography.

An approach was proposed that includes the registration of light data at the microcontroller level and its centralized processing, storage and interpretation in the web environment. All project modules (hardware, program logic, and user interfaces) are open for further expansion.

The Lilka hardware platform, based on ESP32, was chosen as the main technology stack, providing reliable data collection and pre-processing. The backend was implemented in Node.js in combination with PostgreSQL. It allows for scalability and structured data storage. Modularity and openness allow for the gradual integration of new sensors, analytical algorithms, and event processing mechanisms without compromising compatibility.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – програмний інтерфейс застосунків (англ., Application Programming Interface)

CCT – корельована колірна температура (англ., Correlated Color Temperature)

CSV – значення, розділені комами (англ., Comma Separated Values)

EV – значення експозиції (англ., Exposure Value)

IR – інфрачервоний (англ., Infrared)

ISO – міжнародна організація зі стандартизації (англ., International Standards Organization)

UI – користувацький інтерфейс (англ., User Interface)

UX – користувацький досвід (англ., User Experience)

ВСТУП

У сучасних умовах розвитку аналогової та цифрової фотографії, а також досліджень у галузі оптики, зростає потреба у точному вимірюванні та логуванні параметрів світлового середовища. Дана система покликана забезпечити автоматизований збір, збереження та аналіз даних про освітленість, спектральні характеристики та інші фактори, що впливають на якість зображення або функціонування будь-якої системи, залежної від умов освітлення.

Завдяки стрімкому розвитку сенсорної електроніки, мікроконтролерних платформ та розподілених обчислювальних систем відкриваються нові можливості для створення доступних, високоточних та модульних рішень у сфері вимірювання оптичних параметрів середовища. У цьому контексті запропонована система має бути точною та програмно гнучкою. Вона розглядається не як вимірювальний прилад, а як основа для побудови розширюваної екосистеми, здатної інтегруватися у широке коло застосувань – від польових фотодосліджень до освітньої діяльності та наукового експерименту.

Запропонований підхід передбачає комплексне охоплення процесу, включає реєстрацію світлових даних на мікроконтролерному рівні та їхню централізовану обробку, зберігання та інтерпретацію у вебсередовищі. Усі модулі проекту (апаратна частина, програмна логіка та інтерфейси користувача) повинні бути відкритими до подальшого розширення. Це надає системі не лише функціональності, а й дослідницької цінності, як у контексті розвитку алгоритмів аналізу світла, так і для підготовки нових фотографічних практик на основі формалізованих даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис проблеми

Світлове середовище є визначальним чинником у формуванні візуального зображення, як у фотографії, так і в суміжних галузях – кінематографі, машинному зорі, візуалізації даних тощо [1, 20, 22, 27, 29]. Незважаючи на це, у багатьох прикладних сценаріях відсутній систематизований підхід до його вимірювання, аналізу та логування [2]. Проблема ускладнюється суб’єктивністю сприйняття світла, а також обмеженістю стандартних інструментів експонометрії. Вимірювання світлових параметрів часто характеризується недостатньою глибиною аналізу [3]. У фотографії, особливо аналогової, точне відтворення умов експозиції має вирішальне значення для подальшої обробки та аналізу зображень. В умовах відсутності об’єктивних вимірів параметрів світла, фотограф змушений покладатися на інтуїцію або суб’єктивні спостереження. Використання додаткових або вбудованих приладів заміру експозиції або інших корисних параметрів умов зйомки накладає свої обмеження, що унеможливлює системне дослідження причин експозиційних похибок. Нестача структурованих даних про середовище робить майже неможливою побудову моделей взаємодії світла з фотоматеріалом.

Окрім фотографії, завдання точного фіксування параметрів світлового середовища є необхідною умовою у низці суміжних дисциплін, зокрема у комп’ютерному зорі, сценографії, архітектурному світловому дизайні, та експериментальних дослідженнях [4]. У цих галузях світло виступає не лише як умова спостереження, а як активний чинник, що змінює властивості об’єктів, впливає на поведінку систем або є предметом дослідження сам по собі. Без точної фіксації його параметрів (інтенсивності, спектрального розподілу, напрямку) неможливо забезпечити відтворюваність

експериментів, коректність візуалізації або обґрунтованість дизайнерських рішень. Це зумовлює потребу в універсальних методах логування та аналізу світлового середовища, що виходять за межі суто фотографічних задач.

Існуючі засоби вимірювання (експонометри, люксометри тощо) надають миттєві показники, але не завжди забезпечують зручне логування даних у динаміці для подальшого аналізу. Вони орієнтовані на разове використання, не враховують контексту зйомки (відстань, кут, температура навколишнього середовища, час тощо) і не призначені для інтеграції з цифровими системами зберігання. Це звужує спектр їх використання у наукових, освітніх і технічних цілях.

Логування параметрів світлового середовища є важливим елементом процесу для відтворюваності експериментів у візуальних дослідженнях. Без детального журналу змін у світлі неможливо коректно зіставити зображення з умовами їх створення. Це особливо актуально в процесах калібрування, архівації, а також при формуванні баз даних для машинного навчання [5]. Відсутність репрезентативних світлових даних веде до втрати інформаційної цінності знімків.

У контексті вивчення реакції фотоматеріалів, зокрема фотоплівки, на різні спектральні умови, завдання логування стає необхідним для побудови емпіричних або аналітичних моделей. Плівкові матеріали мають різну спектральну чутливість, яка по-різному взаємодіє з джерелами світла різної температури та спектру. Без точного знання умов зйомки неможливо дослідити закономірності зміни кольору, контрасту, насиченості тощо [25].

У цифровій фотографії аналіз світлового середовища дозволяє будувати інтелектуальні системи автоматичного налаштування експозиції [21, 26, 30, 32]. Проте, більшість сучасних алгоритмів оцінки сцени працюють у межах камери, без доступу до зовнішніх фізичних сенсорів. Це створює невідповідність між фізичними характеристиками сцени та цифровими рішеннями. Усунення цього розриву вимагає систематичного збору багатовимірних світлових даних.

Проблема логування також охоплює часовий аспект: освітленість, спектр та інші характеристики можуть змінюватися динамічно, особливо при зйомці на відкритому повітрі або в мінливому середовищі. Разове вимірювання не дозволяє зафіксувати ці зміни, а отже – не дає повної картини світлових умов. Безперервне або періодичне логування є єдиним способом дослідити динаміку освітлення та її вплив на результат.

Важливою складовою задачі є зв'язування параметрів світлового середовища з контекстом зйомки: положенням камери, характеристиками об'єкта, фізичними умовами простору. Ізольовані значення освітленості чи спектру не дають достатньої інформації без просторової прив'язки. Це особливо важливо для наукової фотографії, де метадані мають велике значення для інтерпретації результатів.

У науці та інженерії часто виникає потреба у відтворюваності умов експерименту – зокрема у біології, археології, матеріалознавстві, криміналістиці [5]. Якщо під час візуального спостереження або зйомки не зафіксовано рівень освітлення, температуру кольору, тип джерела світла, то повторити дослід неможливо. Це знижує цінність результатів і унеможливлює верифікацію.

Після накопичення достатньої кількості інформації щодо світлового середовища та конкретних результатів реакції фотоматеріалів, постає задача класифікації та прогнозування, що потребує якісно анотованих наборів даних.

Без достовірної та релевантної інформації про світлові умови глибина моделей машинного навчання обмежується, а результати стають менш точними [5]. Логування світлового середовища сприяє формуванню репрезентативних датасетів та візуалізацій, у яких знімки супроводжуються точними числовими параметрами.

Логування світлового середовища має не лише технічне, а й методологічне значення: воно дає змогу формувати системний підхід до побудови умов зйомки. Замість випадкового або емпіричного підбору

параметрів з’являється можливість розробити типові профілі, шаблони та сценарії зйомки, що підвищує ефективність як творчого, так і дослідницького процесу.

Освітнє значення задачі полягає у розвитку аналітичного мислення у фотографів, дизайнерів та технічних спеціалістів. Замість суб’єктивного сприйняття світла вони отримують змогу працювати з об’єктивними даними, аналізувати і робити висновки на основі числових значень. Це сприяє переходу від інтуїтивної до раціональної фотографії [31, 33].

У сфері автоматизованих систем керування зйомкою або сценами, таких як студійне освітлення, аналіз світлового середовища виступає як джерело зворотного зв’язку для системи, де без чітких даних про реальні умови неможливо розробити ефективні алгоритми керування. Задача логування важлива при побудові замкнених адаптивних систем.

Проблема логування та аналізу світлового середовища є міждисциплінарною, актуальною й методологічно важливою для цілої низки прикладних та дослідницьких напрямів. Вона вимагає розвитку нових підходів, інструментів і стандартів, які дозволять фіксувати світло, задля повноцінно вимірюваної та керованої візуальної інформації.

1.2 Аналіз існуючих рішень

Враховуючи описану в попередньому розділі проблематику (нестабільність спектрального складу освітлення, потреба в контролі експозиції в динамічних або неконтрольованих умовах тощо), виникає необхідність огляду вже наявних технічних засобів.

Проблема логування та аналізу світлового середовища є спільною для широкого кола дисциплін, зокрема для фотографії, сценографії, архітектурного освітлення, комп’ютерного зору. Задачі цих дисциплін потребують фіксування параметрів світлового середовища, їх структуризацію та подальше використання. Серед інструментів, що традиційно

використовуються для цих цілей, можна виокремити чотири основні категорії: люксометри, колориметри, точкові експонометри та інтегральні експонометри. Кожен із цих інструментів виконує специфічну функцію у вимірювальному процесі, водночас демонструючи певні переваги й обмеження, які важливо врахувати в контексті побудови цілісної системи логування та аналізу світлових умов.

1.2.1 Загальні апаратні засоби

Люксметр є фотометричним приладом (рисунок 1.1), що призначений для вимірювання рівня освітленості поверхні в люксах, які відображають щільність світлового потоку на одиницю площі. Основу конструкції пристрою становить фотоелектричний елемент, здатний перетворювати світлову енергію на електричний сигнал з високою точністю лінійної залежності.

Рисунок 1.1 – Сучасний цифровий люксометр

Залежно від конструкції та призначення, діапазон вимірювань люксметрів може варіюватися від сотих часток люкса до десятків тисяч люксів, що дозволяє використовувати їх у широкому спектрі завдань. У промисловості та архітектурі люксметри застосовуються для контролю відповідності освітлення санітарно-гігієнічним нормам на робочих місцях і в громадських приміщеннях.

У фотографії цифрові люксометри допомагають визначити оптимальні параметри експозиції за умов складного або змінного освітлення [23, 24]. Сучасні цифрові люксметри (рисунок 1.1) часто оснащені функцією запису та аналізу даних, що дозволяє здійснювати моніторинг освітленості протягом тривалого періоду часу з можливістю подальшої статистичної обробки результатів.

Колориметр представляє собою спеціалізований оптичний пристрій, призначений для об’єктивного вимірювання кольорових характеристик світла, поверхонь та прозорих середовищ з високою точністю відповідно до колориметричних стандартів.

Функціонування колориметра базується на принципі трихроматичності кольорового зору, при якому будь-який колір може бути представлений як комбінація трьох основних кольорів у певних пропорціях або через координати у стандартизованому колірному просторі (наприклад, CIE XYZ, CIE Lab, RGB) [19].

Типовий колориметр складається з джерела стандартного освітлення (часто з кольоровою температурою 5000K або 6500K, що відповідає стандартному денному світлу), системи світлофільтрів, які моделюють функції відповідності кольорів стандартного спостерігача, фотоелектричних сенсорів та аналітичного блоку для обробки отриманих сигналів. Окрім базових координат кольоровості, сучасні колориметри можуть визначати такі параметри, як колірна температура (K) [15], індекс передачі кольору (CRI), колірні відмінності (ΔE), домінуюча довжина хвилі та чистота кольору.

1.2.2 Фотографічні спеціалізовані прилади

Точковий експонометр є спеціалізованим фотографічним приладом, що забезпечує вимірювання яскравості в обмеженому полі зору (зазвичай 1-5 градусів), що дозволяє фотографам отримувати дані про освітленість конкретних ділянок сцени, а не усередненого значення для всього кадру. Конструктивно точковий експонометр складається з оптичної системи з вузьким кутом огляду [16], світлочутливого елемента, аналітичного блоку та системи індикації, що відображає результати вимірювань у вигляді експозиційного числа (EV), яскравості (кд/м²) або рекомендованих значень діафрагми та витримки.

Принцип функціонування приладу базується на перетворенні світлового потоку від обраної ділянки об’єкта в електричний сигнал з наступним перерахунком у параметри експозиції з урахуванням встановленої чутливості фотоматеріалу (ISO/ASA). На відміну від інтегральних експонометрів, точкові прилади дозволяють здійснювати зональні вимірювання за методом Адамса, при якому різні тональні зони сцени співвідносяться з певними експозиційними значеннями, що забезпечує оптимальне відтворення тонального діапазону [6].

Точкові експонометри особливо ефективні при роботі зі сценами з високим контрастом, де необхідно зберегти деталі як у світлих, так і в темних ділянках зображення. У кінематографі точкові експонометри використовуються для вимірювання яскравості ключових об'єктів сцени та визначення контрастного співвідношення між різними елементами кадру [7]. Професійні моделі точкових експонометрів часто оснащені функцією запам’ятовування кількох вимірів з подальшим усередненням, що дозволяє швидко визначати середню яскравість об'єктів сцени.

Сучасні високоточні точкові експонометри (рисунок 1.2) мають точність вимірювання до 0,1 EV і можуть працювати в умовах надзвичайно низької освітленості (до -5 EV), що зазвичай використовується для

спеціалізованої студійної та технічної фотографії. Інтеграція цифрових технологій дозволяє розширити функціональність точкових експонометрів, доповнивши їх можливостями вимірювання колірної температури, спектрального складу світла та синхронізації з системами студійного освітлення, що значно підвищує ефективність їх використання в професійній фотографії.

Рисунок 1.2 – Професійний цифровий експонометр

Інтегральний експонометр є інструментом фотографії, що здійснює вимірювання середньої яскравості всієї сцени або її значної частини з подальшим перерахунком отриманих даних у параметри експозиції, необхідні для оптимального відтворення зображення. Технічно інтегральний експонометр складається з ширококутової оптичної системи з кутом огляду близько 30-60 градусів, фотоелектричного елемента та аналітичного блоку, що проводить розрахунок експозиційних параметрів з урахуванням встановленої чутливості світлочутливого матеріалу.

Принцип дії пристрою полягає в інтегруванні (усередненні) світлового потоку від різних частин сцени, часто з певним зважуванням (центрально-зважене вимірювання), при якому центральна частина кадру має більший вплив на кінцевий результат.

Історично інтегральні експонометри розвивалися від окремих ручних пристроїв до вбудованих систем вимірювання експозиції в камерах, де вони досі залишаються основним методом автоматичного визначення експозиційних параметрів.

Сучасні інтегральні експонометри використовують матричні або сегментні системи вимірювання, при яких поле зору розділяється на десятки або сотні окремих зон, дані з яких аналізуються з використанням складних алгоритмів та бази даних типових сцен для визначення оптимальної експозиції.

У професійній практиці фотографи часто використовують інтегральні експонометри для швидкого визначення базової експозиції з подальшим коригуванням для конкретних умов зйомки. Технологічно інтегральні експонометри еволюціонували від аналогових пристроїв з механічною індикацією до цифрових систем з мікропроцесорним керуванням та складними алгоритмами аналізу сцени.

Незважаючи на поширення високоточних точкових експонометрів, інтегральні системи залишаються найбільш універсальним засобом вимірювання експозиції для більшості фотографічних завдань завдяки їх здатності швидко надавати збалансовані експозиційні параметри для типових сцен.

Більшість сучасних інтегральних експонометрів обладнані системами корекції для різних джерел світла, що дозволяє враховувати спектральні особливості освітлення (денне світло, вольфрамові лампи, флуоресцентне освітлення тощо) [28]. Також крім апаратних реалізацій, існують і програмні рішення, такі як застосунок LightMeter на мобільний телефон.

1.2.3 Програмний застосунок LightMeter

Застосунок реалізує функціональність портативного світломіра на базі апаратних сенсорів мобільного пристрою, поєднуючи переваги цифрових технологій із методикою використання класичних експонометрів. Застосунок підтримує два режими вимірювання: інтегральний, що використовує вбудований датчик освітленості для оцінки загальної освітленості сцени, та відбитковий, в якому аналізується світло, відбите від об’єкта через камеру смартфона (рисунок 1.3). У першому випадку забезпечується швидке зчитування світлового потоку на площину датчика, а в другому – точніший облік спектральних і просторових особливостей освітлення програмними алгоритмами.

Великий діапазон ISO дозволяє адаптуватися до умов від яскравого денного світла до слабо освітлених інтер’єрів. Користувачам надається можливість вручну коригувати ISO, витримку та діафрагму, а також застосовувати компенсацію експозиції, що важливо при роботі з матеріалами різної світлочутливості або ND-фільтрами. Після вимірювання застосунок виконує розрахунок значення exposure value (EV) за загальноприйнятими формулами [17], відображаючи як рекомендовані параметри камери, так і абсолютні показники освітленості в люксах.

Водночас слід враховувати, що точність інтегрального режиму залежить від кутової чутливості та спектрального діапазону вбудованого датчика смартфона, а відбитковий режим – від характеристик оптичної системи камери. Користувацький інтерфейс побудовано з урахуванням принципів мінімалізму: екран вимірювання відображає лише необхідні показники: lux, EV, рекомендована витримка, а додаткові опції приховано в контекстних меню.

LightMeter є ефективним інструментом, що забезпечує мобільність, гнучкість і відкритий формат даних, роблячи смартфон багатофункціональним світломіром. Його застосування виправдане в першу

чергу для швидкого полевого вимірювання. Проте, прив’язка до сенсорів телефона без зміни апаратного забезпечення є суттєвим недоліком для подальшого розвитку проєкту.

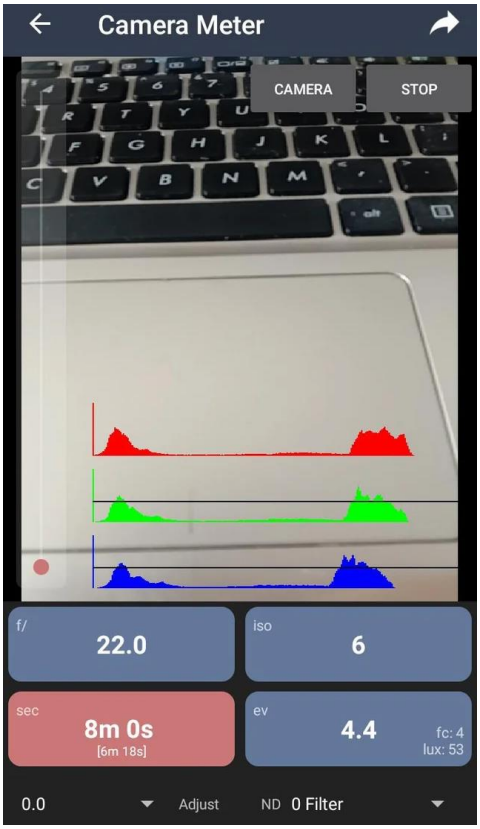

Рисунок 1.3 – Інтерфейс застосунку LightMeter (режим вимірювання камерою)

1.2.4 Переваги та недоліки аналогів

Люксометр – швидке та просте вимірювання загальної освітленості, широкий діапазон чутливості, доступність і низька вартість. Але через відсутність спектральної інформації, неможливість оцінити колірну температуру та напрям, не підходить для аналізу динаміки світла.

Колориметр – точне визначення колірної температури та координат у колірному просторі, вимір CRI та ΔE. Проте висока вартість обладнання, потреба калібрування, проблеми підтримки безперервного логування, складність використання для динамічних змін спектру.

Точковий експонометр – зональне вимірювання працює в умовах низької освітленості, але прибор складний у використанні та має низьку ефективність при динамічній зйомці. Інтегральний експонометр забезпечує усереднення по всій площі кадру, швидке визначення базової експозиції, вбудований у більшість камер, однак характеризується втратою локальних контрастних деталей, а обмежена спектральна й динамічна інформація дають непередбачувані результати в екзотичних умовах.

Найбільш розповсюджені завдання архітектурного світлового дизайну, наукових експериментів чи інженерних досліджень передбачають використання високоточних колориметрів та HDR систем зі складними калібруванням, дорогим обладнанням і закритою екосистемою програмного забезпечення. Хоча ці рішення й надають вичерпну інформацію про спектральні властивості та широкий динамічний діапазон, їхня вартість та інтеграційна складність роблять їх недоступними для щоденних потреб фотографії. У фотоіндустрії існують два прості класи метричних приладів – інтегральні та точкові експонометри, які швидко й інтуїтивно визначають базові параметри експозиції, та вбудовуються безпосередньо в камери або працюють автономно. Втім, ці прилади обмежені одноразовим заміром, не призначені для довготривалого збереження даних і не мають зручних інтерфейсів для експорту результатів у форматах, придатних для глибинного аналізу чи моделей машинного навчання. Різниця між технічними можливостями наукових приладів та простотою фотографічних експонометрів визначає необхідність створення рішення з базовим функціоналом інтегрального експонометра, доповненим цифровим логуванням у зручному виді та аналізом зібраних даних. Такий пристрій водночас має забезпечувати користувачам швидкість і простоту, а також надавати світлові метрики у відкритому форматі для подальшої обробки, побудови кореляцій, тренування нейронних мереж чи інтеграції з апаратними та/або програмними open source платформами та сервісами.

1.3 Постановка задачі

Враховуючи вищеописані обмеження традиційних засобів виникає необхідність розробити модульну систему логування та аналізу параметрів світлового середовища, яка поєднує простоту використання інтегрального експонометра з ефективністю цифрових рішень.

Головним завданням прототипу є перевірка ефективності централізованого збору даних про світло із можливістю подальшої обробки та тренування моделей. Водночас система повинна бути максимально простою в апаратній реалізації, використовувати відкриті апаратні платформи (open source) і забезпечувати інтеграцію з будь якими зовнішніми сервісами. Такий підхід дозволить швидко перевірити базову ідею проєкту та провести подальші експерименти з аналізу світлових метрик. Водночас модульна архітектура створить передумови для поступового розширення функціональності – додавання зонального та спектрального аналізу, роботи з мультисенсорами, інтеграції з іншими сенсорами тощо.

Головним елементом системи є модуль світлометрії, призначений для вимірювання базових фотометричних характеристик: освітленості в люксах та колірної температури в кельвінах [18]. На основі зчитаних показників здійснюється розрахунок значення exposure value (EV) за загальноприйнятими формулами, а також автоматичний підбір оптимальної витримки і діафрагми згідно з заданим ISO. Всі обчислені та необроблені дані записуються в пам’ять пристрою в зручному табличному форматі (CSV), що забезпечує легкий експорт і відкриту структуру рядків і стовпців. Формат CSV обрано як універсальний і сумісний із більшістю аналітичних інструментів і мов програмування. Модуль повинен підтримувати часові мітки (timestamp) з достатньою точністю (мінімум до мілісекунди) для аналізу динаміки освітленості. Завдяки модульності можливе подальше підключення спектрального сенсора або ультрафіолетового/інфрачервоного датчика за потреби.

Другий компонент системи забезпечує надійну передачу та зберігання зібраних світлових метрик із подальшою їх візуалізацією й управлінням конфігурацією пристрою через вебінтерфейс. Після кожного заміру дані автоматично надсилаються на сервер, де вони організовуються в діаграми або таблиці з прив’язкою до міток часу та додаткових контекстних атрибутів.

Інтерфейс користувача надає можливість переглядати зміни освітленості, колірної температури та розрахованого EV у вигляді інтерактивних графіків і теплових карт, а також обирати довільні діапазони для глибинного аналізу. Користувач може застосовувати фільтри за часом, метаданими чи рівнями показників і експортувати вибрані зрізи даних у стандартних форматах для подальшої обробки в статистичних та машинно навчальних середовищах.

Крім можливостей аналітики, вебінтерфейс забезпечує функцію дистанційного налаштування: зміну інтервалу замірів та коригування коефіцієнтів калібрування. Всі зміни конфігурації синхронізуються з пристроєм у режимі реального часу, що дозволяє адаптувати систему до нових умов без фізичного доступу до апаратної частини.

Для забезпечення довгострокового збереження передбачено автоматичне архівування старих записів із можливістю відновлення через вебпанель. Таким чином, програмна частина (клієнт-серверний застосунок) виступає єдиним центром керування даними та налаштуваннями, об’єднуючи збір, зберігання, візуалізацію та конфігурацію в одному зручному середовищі.

У результаті рішення охоплює точну світлометрію з логуванням та повноцінну веб інтеграцію для збереження та візуалізації даних. Кожен із цих компонентів є основою наступних етапів розвитку – зонального та спектрального аналізу, машинного навчання профілів фотоматеріалів і побудови рекомендованих сценаріїв зйомки. Такий структурований підхід забезпечить масштабованість, відкритість та практичну цінність проекту для фотографів на плівку, дослідників і інженерів.

2 ТЕХНОЛОГІЇ РОЗРОБКИ

2.1 Загальний огляд

Розробка системи обох прототипів базується на трьох основних модулях, які утворюють комплексне рішення для світлометрії, зберігання та аналізу даних, а також взаємодії з користувачем.

Перший компонент – це апаратний пристрій, створений на базі мікроконтролера Arduino (перший прототип) або ESP32 з використанням платформи Лілка (другий прототип). Вибір Arduino обумовлений простотою використання та наявністю готових бібліотек під різноманітні задачі. Вибір ESP32 обумовлений високою продуктивністю мікроконтролера, широкими можливостями для підключення датчиків, підтримкою Wi-Fi та Bluetooth, а також зручністю розробки завдяки бібліотеці lilka, що спрощує апаратно-програмну інтеграцію та автоматизацію процесів при переході на другий прототип проекту. Загалом, пристрій відповідає за збір даних, попередню обробку та логування у зручному форматі.

Другий модуль – це серверна частина, реалізована на Node.js. Використання Node.js забезпечує ефективну асинхронну обробку запитів, масштабованість і простоту інтеграції з різноманітними сервісами. Бекенд виконує функції прийому даних з пристрою, їх збереження у базі даних, а також надає API для клієнтського застосунку. Для роботи з даними обрана реляційна система керування базами даних PostgreSQL для надійного зберігання, виконання складних запитів та аналітики. Такий стек дозволяє гнучко керувати даними і реалізовувати складні задачі бізнес-логіки.

Третій модуль – це вебклієнт, розроблений на React. React обрано через його ефективність у створенні інтерактивних інтерфейсів, можливість компонентного кодування та швидкий відгук користувацького інтерфейсу. Вебзастосунок забезпечує відображення зібраних даних, візуалізацію, а

також функціонал налаштування пристрою та системи тригерів. Клієнтська частина орієнтована на простоту і зручність, дозволяє користувачу швидко орієнтуватися у даних і керувати системою в режимі реального часу.

Необхідно забезпечити максимальну модульність і відкритість розробки. Кожна з трьох частин системи розвивається незалежно, взаємодія здійснюється через чітко визначені контракти та протоколи. Це дозволяє легко змінювати або оновлювати компоненти без втрати сумісності. Важливо, що використання популярних та широко підтримуваних технологій забезпечує довгострокову підтримку проекту і відкриває можливості для спільної роботи та розширення.

Особливу увагу приділено простоті інтеграції: пристрій легко підключається до бекенду через REST API, а вебклієнт – до бекенду для відображення і конфігурування. Вся система розробляється з урахуванням принципів відкритого коду, що створює сприятливі умови для спільного розвитку, кастомізації та впровадження нових функцій. Крім того, під час розробки враховується можливість подальшого масштабування – збільшення кількості підключених пристроїв, розширення аналітичних можливостей. Архітектура підтримує горизонтальне масштабування бекенду і бази даних, а також інтеграцію з іншими сервісами через API. Таким чином, загальний підхід полягає в побудові гнучкої, масштабованої та надійної платформи, яка поєднує сучасні апаратні рішення з гнучким програмним забезпеченням і зручним інтерфейсом. Це дає змогу створити інструмент, який не лише вимірює світло, а й дозволяє глибоко аналізувати дані та адаптувати систему під індивідуальні потреби користувача.

2.2 Платформа Lilka

Платформа Лілка (Lilka) [8] – це відкритий апаратно-програмний проект, побудований на базі мікроконтролера ESP32, який поєднує в собі гнучкість, продуктивність і доступність для розробників.

Розроблена українською спільнотою, Лілка призначена для створення портативних пристроїв, таких як DIY-консолі, інструменти для навчання програмуванню та експериментальні проекти (рисунок 2.1).

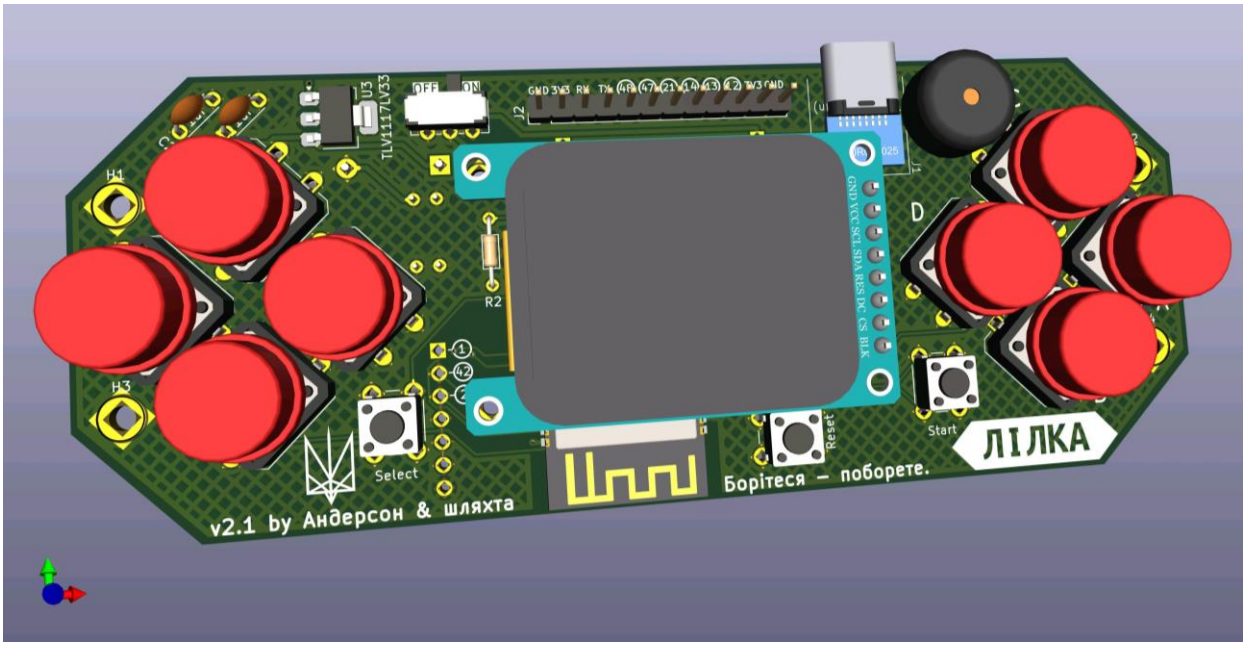

Рисунок 2.1 – Зовнішній вигляд консолі

Завдяки відкритій архітектурі та детальній документації, Лілка стала популярною серед ентузіастів і розробників. Основою Лілки є мікроконтролер ESP32-S3-WROOM-1-N16R8, який має двоядерний процесор з тактовою частотою до 240 МГц. Цей чіп забезпечує високу продуктивність при низькому енергоспоживанні.

Мікроконтролер ESP32-S3 також підтримує Wi-Fi та Bluetooth, що дозволяє легко інтегрувати бездротові функції у проекти. Крім того, використаний у складі платформи Лілка мікроконтролер має вбудовані периферійні модулі, такі як SPI, I2C, UART, що спрощує підключення різноманітних сенсорів і пристроїв.

Лілка оснащена 16 МБ флеш-пам’яті та 512 КБ оперативної, що забезпечує достатній обсяг для більшості застосунків для мікроконтролерів. Плата також має роз’єм для карти microSD, дозволяючи зберігати великі обсяги даних або використовувати її для оновлення прошивки.

В документації рекомендовано для розробки використовувати PlatformIO IDE [10]. А встановивши спеціально адаптовану під пристрій операційну систему KeiraOS, можна писати програми не тільки на C++, але й на Lua або mJS.

Бібліотека lilka також надає API для роботи з апаратними ресурсами, такими як дисплей, кнопки, сенсори тощо. Це спрощує процес розробки та дозволяє зосередитися на логіці програми, не витрачаючи час на низькорівневе програмування. Лілка є універсальною платформою, яка може бути використана в різних сферах. Завдяки своїй гнучкості та функціональності, вона підходить для створення портативних ігрових консолей, освітніх інструментів, пристроїв для збору даних, а також для реалізації IoT-проектів.

Крім того, Лілка може бути використана як основа для розробки прототипів нових пристроїв та технологій. У контексті цього проєкту Лілка є апаратною базою для збору та обробки світлових даних. Завдяки підтримці Wi-Fi та Bluetooth, вона може передавати зібрані дані на сервер для подальшого аналізу. Крім того, можливість підключення різноманітних сенсорів дозволяє адаптуватися під конкретні потреби.

2.3 Зберігання даних

Зберігання даних організовано багаторівнево, що забезпечує як надійність, так і гнучкість доступу. На рівні пристрою дані тимчасово буферизуються у вигляді CSV-файлів, які зберігаються у файловій системі мікроконтролера або на зовнішньому носії. Після цього дані передаються на сервер через Wi-Fi з використанням REST API. На сервері працює API на Node.js, яка відповідає за валідацію, збереження та структурування даних у PostgreSQL. База даних організована навколо логічних одиниць: заміри, профілі пристроїв, конфігурації тощо. Це дозволяє будувати запити для аналітики, фільтрації та візуалізації у вебінтерфейсі. Завдяки збереженню

даних у централізованій БД забезпечується їхній резервний захист, консистентність і можливість колективного доступу. Це надає змогу у майбутньому експортувати дані у хмарні сервіси та додати підтримку альтернативних форматів, зокрема JSON. Такий підхід дозволяє адаптуватися до нових вимог та зберігати сумісність з іншими платформами.

Формат CSV (Comma-Separated Values) використовується в системі як основний спосіб збереження та експорту зібраних світлометричних даних. Це один із найпростіших і найпоширеніших форматів для представлення табличної інформації, який підтримується практично всіма інструментами роботи з даними – від текстових редакторів до спеціалізованого аналітичного програмного забезпечення. Кожен запис у CSV-файлі відповідає одному вимірюванню і містить такі параметри: дата і час, рівень освітленості (люкси), кольорова температура (K), розрахований EV, рекомендована витримка тощо. Такий підхід забезпечує прозорість і доступність для читання навіть без спеціальних програмних засобів.

Дані у форматі CSV легко імпортувати в Excel, Google Sheets, R, Python, MATLAB або бази даних для подальшої обробки. Це особливо важливо на етапі аналізу, створення графіків і підготовки даних до навчання аналітичних моделей. CSV також полегшує процес обміну даними з іншими користувачами або системами. Через просту структуру файл не потребує додаткових бібліотек для парсингу в більшості мов програмування. Крім того, збереження у CSV дозволяє вести журнал всіх вимірювань в архівному вигляді для подальшої перевірки, аудиту чи порівняльного аналізу.

2.4 Огляд сенсорів

Світлові сенсори є основою у проектах, пов’язаних з аналізом навколишнього освітлення, зокрема в системах експонометрії, логування світлового середовища, автоматизації та обробки подій. Головна мета використання світлових сенсорів полягає у перетворенні фізичних параметрів

світла (інтенсивність, спектральний склад, колірна температура тощо) у цифрові дані, придатні для подальшої обробки.

Залежно від технології, сенсори можуть вимірювати загальну освітленість (у люксах), рівень інфрачервоного та ультрафіолетового випромінювання, спектральний склад, або навіть симулювати сприйняття кольору людським оком [9].

Розрізняють сенсори з аналоговим виходом (фотодіоди, фоторезистори) та цифрові (з I2C та SPI інтерфейсами), які надають точніші дані, є більш стабільними та зручними у використанні. Цифрові сенсори також мають вбудовані АЦП та обробку шумів, що дозволяє отримувати більш чисті сигнали навіть за несприятливих умов. У більшості практичних застосувань використовуються цифрові сенсори з I2C, які забезпечують просту інтеграцію з мікроконтролерами, зокрема ESP32.

Важливими параметрами при виборі сенсора є діапазон освітленості, точність, наявність компенсації температури, спектральна чутливість, швидкість вимірювань та можливість калібрування. Деякі сенсори забезпечують лише інтегральні значення (загальна яскравість), інші дають спектральні або колірні компоненти (RGB, XYZ, CIE). Саме такі розширені параметри дозволяють обчислювати кольорову температуру в Кельвінах, коригувати експозицію та будувати повноцінні моделі освітлення.

У контексті фотографії та експонометрії важливими є два типи даних: рівень освітленості в люксах (для визначення EV/експозиції) та спектральний склад світла (для аналізу колірної температури та якості освітлення).

Також важливо враховувати чутливість до інфрачервоного – у деяких випадках це може давати хибну інтерпретацію, тому рекомендовано використовувати IR-фільтр або відповідну компенсацію в прошивці.

Важливим критерієм вибору сенсору є стабільність вимірювання у змінних умовах при природному світлі, штучному, флуоресцентному та змішаному освітленні. Сенсор повинен мати стабільну поведінку у всьому діапазоні вимірювання, від тіні до яскравого сонячного світла, забезпечуючи

лінійність вимірювань. Часто сенсори потребують індивідуального калібрування, оскільки заводські значення можуть бути недостатньо точними або адаптованими лише до певного обладнання.

У даному проекті основна мета сенсорної підсистеми полягає у забезпеченні збору даних, на основі яких буде вестися логування світлових умов та здійснюватися обчислення рекомендованих параметрів зйомки. Це передбачає точність, повторюваність та інтерфейсну зручність. Наявність кількох сенсорів дозволяє реалізувати комбінований підхід: один сенсор працює як базовий люксметр, другий для оцінки кольору, третій як спектрометр для подальшої побудови цифрових профілів плівки.

Розширена інформація про спектр дозволяє аналізувати освітлення не лише з точки зору яскравості, а й з урахуванням якості – наприклад, «холодне» чи «тепле» світло. Вимір кольорової температури є важливим для побудови точних експозиційних моделей. Таким чином, світлові сенсори в системі логування – це вимірювальні пристрої, які є базовим компонентом для цифрової реконструкції світлового середовища.

2.4.1 Сенсор BH1750

Сенсор BH1750 [10] (модуль GY-302) – це цифровий датчик освітленості з вбудованим АЦП, що безпосередньо видає значення освітленості (люкс). Він працює в діапазоні приблизно 1…65 535 люкс, а за оптимізації часу виміру може виявляти до ~100 000 лк. Споживання енергії дуже низьке – приблизно 0,12 мА в активному режимі. У сенсора BH1750 малий вплив інфрачервоного випромінювання, тому він має спектральну чутливість, наближену до людського ока. Датчик видає дані по I2C як кількість люкс, без необхідності додаткових розрахунків. Його точність за специфікацією складає близько ±20%.

Серед переваг слід зазначити просте використання (готовий вихід у люксах), дуже низьке енергоспоживання, великий динамічний діапазон (від

слабкого до яскравого світла). Виробник заявляє мінімальний вплив ІЧ-складової, що полегшує вимір «людино-зорового» освітлення [10]. Сенсор дешевий і широко підтримується бібліотеками для Arduino/ESP32.

Серед недоліків – відносно мала заявлена точність (до ±20%) і роздільна здатність, обмежена 16 бітами (похибка може сягати декількох люкс при низьких значеннях). BH1750 не дає інформації про колір світла. Також він може перенасичуватися при екстремально яскравому світлі без додаткового калібрування часу експозиції.

BH1750 – це простий та енергоекономний датчик для вирішення задач регулювання яскравості та обчислення EV відповідно до ступеню освітлення. Однак він спроможний забезпечити лише вимір освітленості, без колірної інформації. Для задач простого інтегрального експонометра його точності зазвичай може вистачити, але вимоги точності в ±1 лк він не забезпечує.

2.4.2 Сенсор TSL2591

Сенсор TSL2591 [11] – високочутливий цифровий датчик освітленості з дуже великим дінамічним діапазоном. Він має два фотодіоди: один вимірює інфрачервоне випромінювання, інший – повний спектр (видимий + ІЧ). За допомогою програмованого посилення і більшому часу інтеграції TSL2591 може працювати при надзвичайно слабкому освітленні (~0.188 мклк) до ~88 000 лк. Модуль споживає лише ≈0.4 мА при активному вимірі і <5 μA у режимі сну. Через підтримку ІЧ спектру та з урахуванням повної площини можна розрахувати «видиму» освітленість, віднімаючи ІЧ, та обчислити освітленість за формулою [12]. Adafruit надає бібліотеку з готовою функцією calculateLux(full, ir), що дає значення освітленості на основі обох каналів.

Серед переваг TSL2591 можна виділити широкий діапазон чутливості (від темряви до яскравої сонячної погоди), мінімальне енергоспоживання при вимірах і в режимі очікування, простоту інтеграції (використання I2C та готових бібліотек для Arduino/ESP32). Однак, TSL2591 не розрізняє кольори,

він дає лише сумарне та ІЧ-випромінювання. Щоб уникнути насичення, потрібно вибирати відповідні налаштування gain/time (бібліотека повідомляє «Invalid data» при переповненні).

Датчик TSL2591 може використовуватися для побудови високоточних світлозалежних приладів з широким діапазоном та для обчислення EV; проте у задачах, що вимагають колірної температури або спектру, доведеться доповнювати його іншим сенсором.

2.4.3 Сенсор AS7262

Сенсор AS7262 [13] – це 6-канальний спектральний датчик видимого світла. Він має 6 вузькоспектральних каналів з піками на 450, 500, 550, 570, 600 та 650 нанометрів, де ширина півмаксимуму ~40 нм). Сенсор видає значення для кожного каналу (16-біт), які можна отримати по I2C як raw data або калібровані числа. Є вбудований індикатор температури чіпа і власний драйвер для LED-підсвічування. У заводській прошивці сенсор калібрований на стабільні температурні умови, тому його результати мінімально залежні від часу експлуатації. З переваг можна виділити високу спектральну роздільну здатність, можливість обчислити корельовану колірну температуру за результатами каналів, використання інтерфейсу I2C або UART і наявність готової бібліотеки. Вбудовані фільтри гарантують стабільність даних при зміні зовнішньої температури.

Ціна AS7262 значно вища за аналоги. Деякі технічні обмеження, як конверсія даних, що займає час (щонайменше ~3 мс при повному наборі каналів), можуть стати вирішальними при виборі сенсора. Також споживання струму модулем з підсвічуванням (LED flash) може сягати десятків міліамперів, а в умовах насиченого світла деякі канали можуть вимагати короткого часу інтеграції (integration time), що треба налаштовувати вручну.

Отже, AS7262 – найбільш складний сенсор із спектральним покриттям видимого діапазону. Він забезпечує високу деталізацію спектру і вимагає

мінімальних допущень при обчисленні CCT. Проте для звичайного експонометра цей сенсор надмірний і складний у використанні. При комбінації функцій експонометра та адаптивних тригерів AS7262 може замінити інші лише якщо потрібна максимальна точність спектрального аналізу (наприклад, для спектроскопії матеріалів).

2.4.4 Сенсор TCS34725

Сенсор TCS34725 [14] (рисунок 2.2) – це RGB-датчик світла з ІЧ-фільтром і вбудованим білим світлодіодом (4150 K) для підсвічування. Він містить 4 фотодіоди (червоний, зелений, синій та clear) за ІЧ-обмежувальним склом. Тому цей модуль вимірює не лише яскравість, а й спектральний склад видимого світла. Динамічний діапазон сенсора виробник вказує як 3 800 000:1. За рахунок регульованого посилення і часу інтеграції TCS34725 здатен працювати від темних приміщень до досить яскравих умов. Для точного визначення колірної температури рекомендується використовувати інтеграційні числа кратні 50 мс.

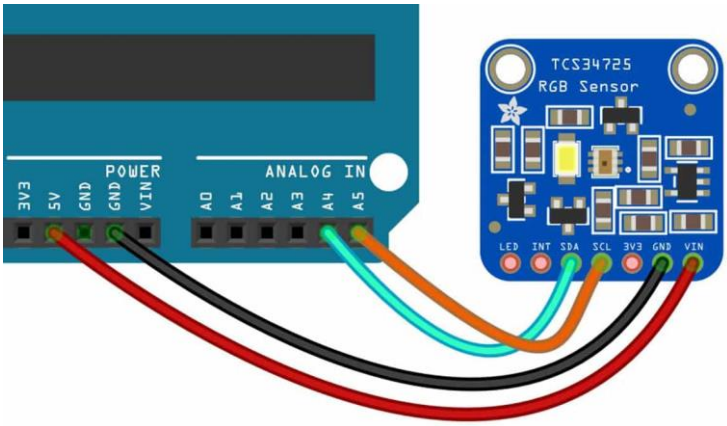

Рисунок 2.2 – Типове підключення сенсора TCS34725 до плати Arduino

За допомогою опублікованих формул [17] з сирих даних RGB і Clear можна обчислити і значення освітленості, і колірну температуру. Серед переваг TCS34725 можна визначити обчислення корельованої колірної

температури, наявність вбудованого світлодіоду, який дозволяє підсвітлювати об’єкт для спектрометрії (проте, це може бути навпаки надлишковим в межах фотографічних задач), широкий динамічний діапазон і можливість гнучкого налаштування підсилення та інтеграційного часу. Ці характеристики роблять TCS34725 універсальним для використання в різних умовах.

Активний споживаний струм модуля невеликий (~0.65 мА за стандартних налаштувань, а користувацькі бібліотеки (Adafruit і альтернативні) дозволяють легко інтегрувати сенсор з ESP32 і отримувати готові величини (RGB, люкс, CCT). Проте, сенсор має нижчу розрядність на канал (16 біт) і потребує правильного налаштування, інакше можливе переповнення або низька чутливість.

Слід зазначити, що у більшості бібліотек переведення в люкси та колірну температуру виконане спрощено, тому похибка для освітленості може сягати декількох люкс. Також за умов ввімкненої підсвітки модуль споживає додатково ≈20 мА (струм світлодіода), а бажана точність ±1 лк досяжна лише при ретельному калібруванні і оптимальному виборі умов виміру.

TCS34725 – найліпший сенсор серед розглянутих для отримання даних щодо освітлення (значення люкс та колірної температури). Він одночасно вимірює інтенсивність і відтінок світла, що важливо для задач, які вирішує дана розробка. Хоча він має обмеження через переповнення при довгих інтеграціях, це можна налаштувати вручну або за програмним алгоритмом.

3 АПАРАТНО-ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Прототип на Arduino

У межах дослідницького етапу розробки системи логування світлового середовища для аналогової фотографії було створено базовий функціональний прототип на основі мікроконтролера Arduino (рисунок 3.1).

Прототип на Arduino мав на меті реалізацію базових принципів взаємодії між сенсорною частиною системи, інтерфейсом користувача та блоком відображення даних. Його функціональне призначення полягало у тестуванні основних принципів, пов’язаних із збором, обробкою та виведенням параметрів, що використовуються для коректної експозиції під час фотографування на плівку. Створення такого прототипу дало змогу сформувати чітке уявлення про вимоги до сенсорних компонентів системи, обґрунтувати вибір формату взаємодії користувача з пристроєм, а також розробити первинну логіку обчислень експозиційних параметрів у реальному часі.

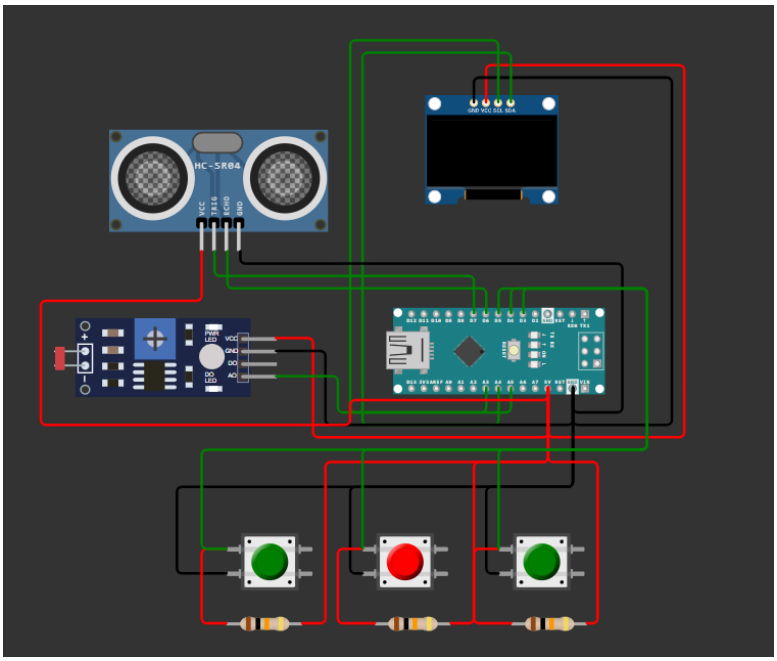

Рисунок 3.1 – Схема прототипу в імітаційному середовищі Wokwi

Робота прототипа організована у вигляді циклічної системи опитування сенсорів з паралельним оновленням візуального інтерфейсу користувача та серійним виведенням діагностичних даних (рисунок 3.2). Його програмна структура ґрунтується на періодичному виконанні основного циклу, в межах якого здійснюється обробка стану кнопок управління, зчитування поточних значень освітленості, кольорової температури та відстані до об’єкта, розрахунок рекомендованої тривалості витримки та оновлення відображуваної інформації.

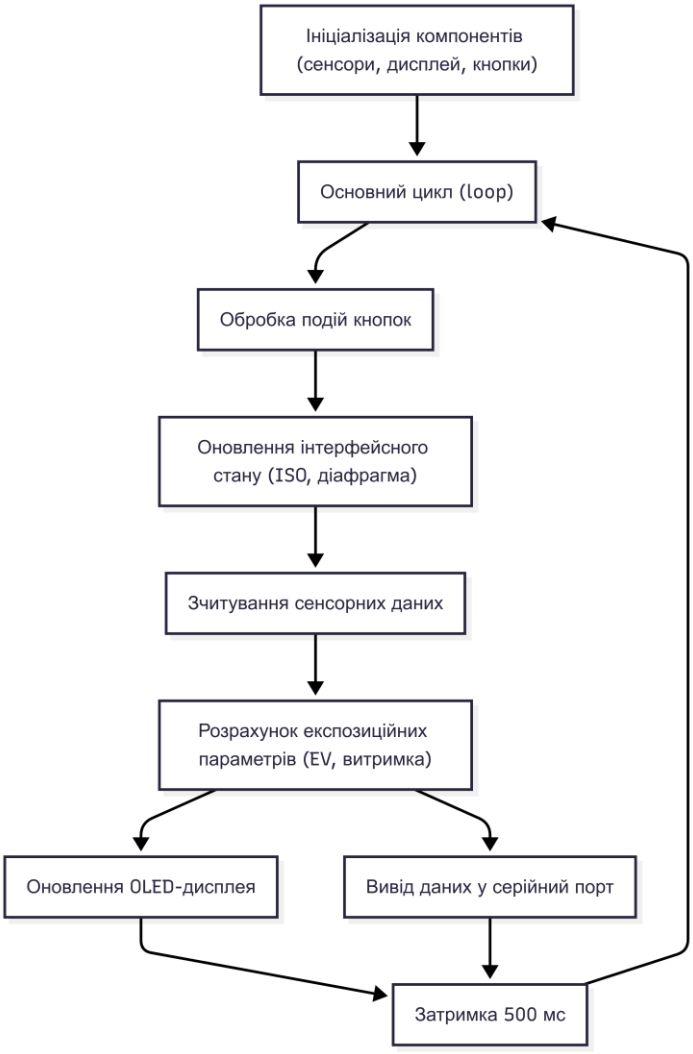

Рисунок 3.2 – Алгоритм роботи прототипу на Arduino

На першому етапі кожної ітерації основного циклу система опрацьовує сигнали від трьох кнопок управління. Ці кнопки призначені для взаємодії

користувача з інтерфейсом та зміни налаштувань, зокрема вибору чутливості ISO та діафрагми. Логіка обробки подій абстрагована у вигляді модуля інтерфейсу, який визначає, чи були натискання на відповідні елементи керування, і відповідно оновлює стан внутрішніх змінних.

Після завершення обробки подій користувача система переходить до зчитування параметрів світла. З фотометричного сенсора отримується значення освітленості в люксах, яке переводиться до експозиційних одиниць (EV), що в подальшому використовується для розрахунку рекомендованої витримки. Додатково зчитується значення корельованої кольорової температури (CCT), яке може слугувати непрямим індикатором типу джерела світла, а отже – бути релевантним для подальшого аналізу якості експозиції. Водночас ультразвуковий сенсор вимірює відстань до об’єкта зйомки, що дозволяє, за необхідності, коригувати експозиційні параметри відповідно до закону обернених квадратів або для інших задач, таких як оцінка зони різкості.

Зібрані значення використовуються для розрахунку рекомендованої витримки. Розрахунок базується на класичній експозиційній формулі, що пов’язує освітленість, чутливість плівки та значення діафрагми з параметром витримки. При цьому значення ISO та діафрагми обираються вручну користувачем, а розрахунок здійснюється автоматично на основі поточного рівня освітленості.

Результати обчислень, а також вхідні параметри виводяться на OLED-дисплей, що забезпечує зворотний зв’язок для користувача. На екрані одночасно відображається низка параметрів: позначене поточне поле, вибране значення, ISO, діафрагма, розрахована витримка, виміряна відстань, EV та CCT. Такий формат подання інформації дозволяє оператору швидко оцінити контекст середовища та скоригувати налаштування, що забезпечує адаптивну поведінку системи при зміні умов зйомки.

Після завершення візуального оновлення всі зібрані параметри також надсилаються до послідовного порту, що слугує як інструментом відладки,

так і засобом для подальшого збору даних у рамках більш комплексної системи. Затримка між ітераціями циклу становить 500 мс, що забезпечує достатню частоту оновлення без надмірного навантаження на ресурси мікроконтролера.

Розроблений прототип виконує роль тестового стенду для оцінки працездатності основних компонентів та алгоритмів системи логування світлового середовища. Його архітектура реалізує базові функції зчитування фізичних параметрів, обробки взаємодії з користувачем, адаптивного обчислення експозиційних характеристик та виведення інформації у реальному часі. Хоча прототип не передбачає збереження цих даних або дистанційної взаємодії, його реалізація є важливою передумовою для переходу до більш складних архітектур, зокрема з використанням ESP32-платформи та мережевих сервісів, що стали основою наступних етапів розвитку системи.

3.1.1 Модуль Button

Модуль Button.h (додаток Б.1.1) реалізує базову функціональність для роботи з кнопками в контексті вбудованих систем, зокрема для мікроконтролерів сімейства Arduino. Його основне призначення полягає в абстрагуванні низькорівневої обробки сигналів від фізичної кнопки, а також у реалізації механізмів фільтрації паразитних спрацювань, викликаних механічними коливаннями контактів (так званий debounce). Така абстракція дозволяє значно спростити логіку основної програми, а також забезпечити стабільну та передбачувану поведінку інтерфейсу користувача.

З точки зору архітектури, клас Button має закриті поля, що містять інформацію про номер цифрового піну, до якого підключена кнопка (pin), поточний логічний стан (state), фіксовану затримку для debounce (debounceDelay), а також часову мітку останнього переходу стану (lastDebounceDelay). Конструктор класу приймає як обов’язковий параметр

номер піну, а також необов’язковий параметр, що задає тривалість debounce-інтервалу в мілісекундах (за замовчуванням – 10 мс). Під час ініціалізації внутрішній стан кнопки встановлюється у логічно неактивний рівень, що відповідає стандартному для конфігурації INPUT_PULLUP підходу.

Метод begin() призначений для одноразової ініціалізації кнопки. Він встановлює режим піну як INPUT_PULLUP, що забезпечує внутрішнє підтягування до логічної одиниці. Така конфігурація є типовою для кнопкових інтерфейсів, оскільки забезпечує надійне визначення неактивного стану навіть при розімкнутому контакті.

Функція isPressed() повертає логічне значення true, якщо кнопка перебуває в активному стані, тобто замкнена на землю. Ця функція не враховує debounce, і використовується у випадках, коли потрібне лише поточне миттєве значення. Більш складні методи wasPressed() та wasReleased() реалізують логіку виявлення моменту натискання та відпускання кнопки відповідно. Обидва методи ґрунтуються на аналізі поточного логічного стану піну, порівнянні його з попереднім станом, а також обчисленні часової різниці між подіями з урахуванням debounce-затримки. Такий підхід дозволяє уникнути багаторазових спрацювань під час одного фізичного натискання, що часто виникає внаслідок контактних коливань, характерних для механічних перемикачів.

Метод wasPressed() активується лише тоді, коли відбувся перехід з логічного високого рівня (стан спокою) до низького (стан натискання), і при цьому з моменту останнього такого переходу пройшло більше часу, ніж вказано у debounceDelay. У випадку спрацювання, стан кнопки оновлюється, а також зберігається нова часова мітка. Аналогічно працює метод wasReleased(), однак він реагує на перехід у зворотному напрямку – від натиснутої кнопки до відпущеної.

Дана реалізація забезпечує мінімально необхідний функціонал для роботи з механічними кнопками в умовах мікроконтролерного середовища. Вона не потребує зовнішніх бібліотек, має низький рівень споживання

ресурсів, і може бути безпосередньо масштабована у випадках збільшення кількості кнопкових елементів у системі. Завдяки чіткому поділу логіки натискання, утримання та відпускання, клас Button є придатним для інтеграції в більш складні інтерфейсні модулі з реактивною логікою.

3.1.2 Модуль Display

Модуль Display (додаток Б.1.2) відповідає за візуалізацію параметрів системи на графічному OLED-дисплеї розмірами 128×64 пікселі з інтерфейсом I2C, побудованому на базі контролера SSD1306. Його функціональне призначення полягає у формуванні компактного та інформативного графічного інтерфейсу, який у реальному часі відображає експозиційні параметри, отримані від сенсорних підсистем, а також забезпечує зворотний зв’язок користувачу щодо поточного стану вибраних або редагованих полів.

На рівні архітектури клас Display є похідним від бібліотечного класу Adafruit_SSD1306, що реалізує базову функціональність роботи з контролером дисплея. Оскільки батьківський клас надає низькорівневі методи для роботи з пікселями, текстом та графікою, Display розширює цю функціональність, додаючи методи стилізації виводу та специфічну логіку відображення параметрів фотозйомки. У конструкторі Display() відбувається ініціалізація об’єкта з використанням стандартних параметрів: фіксовані розміри дисплея (128×64), вказівка на шину Wire (I2C) та відсутність піну скидання (параметр OLED_RESET = -1). Це відповідає типовій конфігурації, яку підтримують комерційно доступні модулі на базі SSD1306. Метод begin() здійснює ініціалізацію апаратної частини дисплея, перевіряючи, чи пройшло коректне підключення, після чого очищає екран і оновлює вміст буфера відображення. У разі невдачі ініціалізації виконується зупинка виконання програми через нескінченний цикл, що є типовим підходом у системах без операційної системи.

Метод draw забезпечує відображення на екрані всіх основних параметрів, що характеризують поточну експозиційну ситуацію. Метод приймає аргументи, що включають:

- marked та selected – параметри, які позначають відповідно підсвічене (активне) та редаговане поле користувацького інтерфейсу;
- iso, aperture, shutter – стандартні експозиційні параметри;
- distance, ev, cct – додаткові вимірювання з сенсорів.

Візуалізація здійснюється з використанням спеціалізованих методів стилізації. Метод drawBoldText() виводить текст з імітацією «жирного» шрифту шляхом багаторазового накладення тексту з невеликим зсувом по координатах. Такий підхід дозволяє реалізувати стилістичне виділення в умовах обмеженої типографіки в бібліотеці Adafruit GFX, яка не підтримує зміну жирності шрифтів.

Метод drawUnderlinedText() реалізує підкреслення тексту шляхом додавання горизонтальної лінії під виведеним написом. Для цього виконується розрахунок ширини тексту у пікселях на основі кількості символів та фіксованої ширини символу шрифту. Такий механізм використовується для позначення редагованого поля в інтерфейсі, що сприяє візуальній орієнтації користувача.

У межах самого методу draw() параметри відображаються у визначеному регіоні екрана. Для кожного поля спершу перевіряється, чи воно поточне (selected) або активне (marked), після чого викликається відповідна функція стилізації – або drawUnderlinedText, або drawBoldText з відповідним рівнем жирності.

Наприклад, ISO виводиться у верхньому лівому куті, а діафрагма – праворуч. Значення витримки, дистанції, експозиційної цінності (EV) та корельованої кольорової температури (CCT) розміщуються у двох нижніх рядках дисплея.

Слід зазначити, що в усіх випадках виведення числових значень доповнюється текстовими одиницями виміру (наприклад, cm, EV, K), що

зберігаються у пам’яті PROGMEM для зменшення використання оперативної пам’яті (SRAM). Це дозволяє вивільнити ресурси мікроконтролера для інших компонентів системи.

Завершальним етапом у методі draw() є виклик display(), який передає підготовлений буфер на фізичний дисплей. Завдяки цьому забезпечується оновлення екрану лише після повного завершення малювання всіх компонентів, що мінімізує мерехтіння й артефакти.

У підсумку, модуль Display реалізує структуровану систему виводу інформації з підтримкою динамічного форматування та інтеграції з логікою інтерфейсу користувача. Його реалізація спирається на можливості апаратного дисплея та розширює базову бібліотеку Adafruit, забезпечуючи гнучкий та інформативний графічний інтерфейс для взаємодії з системою.

3.1.3 Модуль LightSensor

Модуль LightSensor (додаток Б.1.3) відповідає за отримання та попередню обробку фотометричних і колориметричних даних з датчика кольору TCS34725. Він реалізує функціональність зчитування освітленості, розрахунку експозиційної цінності (EV), визначення корельованої кольорової температури (CCT), а також оцінки рекомендованої витримки для заданих параметрів ISO та діафрагми.

Клас LightSensor є похідним від бібліотечного класу Adafruit_TCS34725, що надає базовий інтерфейс до апаратного сенсора через шину I2C. У конструкторі LightSensor() викликається базовий конструктор з параметрами integrationTime та gain, що вказує сенсору працювати з часом інтеграції 101 мс та одиничним коефіцієнтом підсилення (підсилення немає). Обрані параметри забезпечують компроміс між швидкістю зчитування і точністю вимірювання. Встановлене значення часу інтеграції дозволяє датчику накопичити достатню кількість фотонів, зменшуючи вплив шуму в умовах слабкого освітлення, без суттєвого зниження частоти оновлення.

Метод begin() відповідає за ініціалізацію датчика. Він викликає метод базового класу для перевірки наявності пристрою на шині I2C. У разі успішного виявлення сенсора виводиться підтвердження у консоль, а при невдачі – виконання зупиняється у нескінченному циклі.

Метод getEV() реалізує розрахунок значення експозиції (Exposure Value), яка у фотографії використовується для характеристики загального рівня освітленості сцени. EV визначається на основі розрахованого значення освітленості, яке повертає функція calculateLux(), що є частиною бібліотеки Adafruit. Цей параметр переводиться у двійковий логарифм і масштабується відповідно до стандарту EV при ISO 100.

Значення освітленості попередньо нормується через поділ на константу, що дорівнює 2.5, яка в цьому контексті виконує функцію емпіричного коефіцієнта калібрування – це дозволяє адаптувати модель до реальних умов та характеристик оптичної системи (наприклад, поглинання світла сенсором, оптичним фільтром тощо).

Метод getCCT() реалізує розрахунок корельованої кольорової температури на основі поточних значень червоного, зеленого та синього каналів. Для цього використовується бібліотечна функція calculateColorTemperature(), яка імплементує апроксимацію залежності між співвідношеннями кольорових компонентів і колірною температурою світла. CCT є корисним параметром для оцінки спектрального складу джерела світла, що, у свою чергу, може бути використано для вибору відповідної плівки або фільтра, зокрема у контексті аналогової фотографії.

Метод calculateShutter() виконує обчислення рекомендованої витримки за заданих значень EV, ISO та діафрагми. Формула, яку використовує метод, базується на класичному співвідношенні експозиційного трикутника. Дане співвідношення дозволяє адаптувати витримку до зміни освітлення або параметрів об’єктиву, зберігаючи постійний експозиційний рівень.

Метод printToSerial() призначений виключно для діагностики або калібрування системи. Він виводить у консоль необроблені значення каналів

червоного, зеленого, синього та прозорого (clear) компонентів, а також обчислене значення освітленості. Це дозволяє розробнику спостерігати за стабільністю та точністю вимірювань, а також оцінити чутливість сенсора в умовах змінного освітлення. Метод є обмеженим у використанні, оскільки не повинен застосовуватися у фінальному режимі роботи системи через високі витрати часу на послідовну передачу даних.

Таким чином, модуль LightSensor інкапсулює фотометричну функціональність системи, забезпечуючи одночасно доступ до абсолютних і спектральних характеристик середовища. Його реалізація є тісно пов’язаною з експозиційною логікою і формує основу для адаптивної роботи всієї системи в умовах природного або штучного освітлення.

3.1.4 Модуль Ultrasonic

Модуль Ultrasonic (додаток Б.1.4) реалізує функціональність вимірювання відстані до об’єкта за допомогою ультразвукового датчика з двома цифровими каналами: trigPin для ініціалізації імпульсу, та echoPin для прийому сигналу. Клас абстрагує низькорівневу взаємодію з апаратним компонентом і надає інтерфейс для отримання як одиничного, так і усередненого (відфільтрованого) значення відстані в сантиметрах. Подібна інкапсуляція є доцільною для використання в реальному часі в контексті логування, де потрібно зменшити вплив випадкових флуктуацій при вимірюваннях.

У конструкторі класу параметризовано задаються номери пінів trigPin та echoPin, а також кількість вимірювань, які буде використовувати метод фільтрації. Всі значення зберігаються у приватних полях класу.

Метод begin() виконує ініціалізацію пінів мікроконтролера. Зокрема, trigPin встановлюється як вихід, а echoPin як вхід. Одразу після цього на trigPin подається низький рівень, що гарантує початкову неактивну конфігурацію. Цей крок є важливим, оскільки ультразвуковий сенсор очікує

саме на керований імпульс для запуску вимірювання та неправильна ініціалізація може призвести до хибних зчитувань.

Основна функціональність міститься у методі getDistanceCM(), який реалізує одне ультразвукове вимірювання. Алгоритм виконується в кілька етапів. По-перше, на trigPin подається короткий імпульс високого рівня тривалістю 10 мікросекунд. Це відповідає специфікації більшості стандартних ультразвукових модулів типу HC-SR04. Після цього затримка в 500 мікросекунд дозволяє стабілізувати систему перед прийомом сигналу. Далі очікується зміни стану echoPin: датчик утримує високий рівень на цьому піні, поки не отримає відлуння. За допомогою функції micros() вимірюється тривалість проходження імпульсу в обох напрямках (від передавача до об'єкта та назад), після чого розраховується значення відстані у сантиметрах.

Додатково впроваджено механізм обмеження часу очікування (15 мс), що запобігає зависанню в разі, якщо сигнал не повертається – наприклад, через надто далеку або поглинаючу поверхню. У таких випадках функція повертає обмежене значення (150 см), що інтерпретується як максимальна коректна межа вимірювання в умовах поточної реалізації.

Метод getFilteredDistanceCM() реалізує просту фільтрацію шляхом усереднення результатів кількох послідовних вимірювань. Такий підхід дозволяє зменшити вплив випадкових похибок, спричинених зовнішніми перешкодами, варіативністю відбиття сигналу чи тимчасовою нестабільністю в електроживленні. Кількість вимірювань визначається параметром measurementsCount, який задається під час створення об’єкта. Між окремими вимірюваннями встановлено затримку 10 мс, що забезпечує стабільність і запобігає накопиченню залишкових сигналів від попереднього імпульсу.

У контексті загальної системи логування світлового середовища модуль Ultrasonic є допоміжним інструментом. Він надає інформацію про відстань до об’єкта, що може бути використано для фільтрації кадрів з некоректною фокусною площиною. Крім того, значення відстані можуть бути корисними для кореляції з іншими сенсорними даними – наприклад, для

моделювання зміни експозиційної цінності залежно від відстані до джерела світла, або для побудови профілів розподілу освітлення в просторі.

3.1.5 Модуль UserInterface

Модуль UserInterface (додаток Б.1.5) виконує роль логічного контролера інтерфейсу користувача в системі логування світлового середовища. Його основна функція полягає в керуванні процесом навігації та редагування параметрів експозиції за допомогою трьох фізичних кнопок (вгору, вниз, підтвердити). Архітектура модуля побудована навколо внутрішнього стану, що відображає поточне положення курсора marked та активний (редагований) параметр selected. Таким чином, клас інкапсулює як модель станів взаємодії, так і конкретні значення параметрів, представлені індексами в масивах допустимих значень.

Клас містить три масиви фіксованих значень: ISO_VALUES, APERTURE_VALUES і SHUTTER_VALUES. Кожен з них включає типовий набір параметрів, що використовуються у фотографічній практиці. Перший елемент у кожному масиві має значення 0, що зарезервоване як службове. Додатково кожен масив має відповідну змінну _SIZEOF, що забезпечує правильну обробку граничних індексів.

Управління інтерфейсом ґрунтується на двох параметрах:

- marked – позначає поточний параметр, над яким наведено курсор;
- selected – позначає активне поле, що перебуває в режимі редагування.

Тип FL_Parameter, визначений як enum, задає допустимі значення цих станів: ISO, діафрагма, витримка, а також FL_NONE – стан без активного редагування. У конструкторі класу встановлюється початковий стан: курсор знаходиться на параметрі ISO, редагування відключено (selected = FL_NONE), а всі індекси параметрів мають значення 3, що відповідає типовим початковим значенням (ISO 100, F/2.8, 1/250 с).

Метод handleUI() є центральним елементом модуля. Він приймає як вхід логічні значення, що відповідають спрацюванню кнопок (вгору, вниз, підтвердження). Подальша поведінка залежить від поточного режиму – навігація (selected == FL_NONE) або редагування конкретного параметра.

У стані FL_NONE натискання кнопок «вгору» і «вниз» змінює значення marked за фіксованим циклічним порядком. Наприклад, натискання «вгору» перемикає з ISO на діафрагму, потім на витримку, плівку, і знову на ISO. Ця логіка реалізована у вигляді вкладених тернарних операторів, що забезпечують компактну реалізацію. Натискання кнопки підтвердження (buttonAccept) переводить поточний marked у selected, тим самим активуючи режим редагування.

Після активації редагування інтерфейс дозволяє змінювати значення поточного параметра (ISO, діафрагма або витримка) за допомогою кнопок «вгору» і «вниз». Обробка індексів виконується засобами допоміжної функції getNextIndex(), яка реалізує циклічний перехід між елементами масиву. Натискання кнопки підтвердження завершує редагування і повертає стан selected у FL_NONE.

Методи getISO(), getAperture() та getShutter() повертають поточне значення відповідного параметра на основі індексу. Це дозволяє зовнішнім модулям отримувати дані для подальшого використання. Методи getMarked() і getSelected() забезпечують доступ до логіки інтерфейсу, що дозволяє реалізовувати динамічне візуальне виділення поточного або редагованого параметра на екрані.

Модуль UserInterface реалізує мінімалістичний, однак структурований підхід до взаємодії користувача з системою. Його перевагою є чітке розмежування режимів взаємодії при навігації та редагуванні, що робить логіку передбачуваною. Архітектура є масштабованою і дозволяє легко розширити інтерфейс новими полями (наприклад, вибір типу плівки, компенсації експозиції тощо). У поєднанні з модулем Display, цей клас формує завершений цикл взаємодії з системою.

3.1.6 Допоміжні модулі

Модуль I2C_Scanner (додаток Б.1.6) є допоміжним інструментом для діагностики шини I2C. Його основна функція – виявлення підключених пристроїв шляхом перебору всіх можливих адрес у допустимому діапазоні (від 0x01 до 0x7E). Клас не має стану: метод scan() ініціалізує I2C-шину, а потім послідовно надсилає запити на кожну адресу. Якщо пристрій відповідає (тобто Wire.endTransmission() == 0), його адреса виводиться через серійний порт у шістнадцятковому форматі. Цей модуль застосовується винятково на етапах розробки та налагодження – зокрема для верифікації підключення сенсорів, виявлення конфліктів адрес або визначення адрес нових компонентів. Він не призначений для використання в основному циклі програми, але значно прискорює інтеграцію периферії на етапі створення прототипу. У системах на кшталт Arduino з обмеженим обсягом оперативної пам’яті контроль за її використанням є особливо важливим. Тому після ініціалізації низки модулів (сенсори, дисплей, кнопки) пам’ять може бути майже вичерпано – особливо через використання OLED-дисплея, що резервує близько 1 КБ буфера. Це підвищує ризик нестабільної роботи. Для діагностики таких ситуацій застосовується бібліотека MemoryFree, яка дозволяє вивести обсяг вільної пам’яті через freeMemory(). У режимі відладки це допомагає вчасно виявити дефіцит ресурсів. Якщо залишок SRAM критично малий, доцільно оптимізувати код: виносити строки в PROGMEM, зменшувати кількість глобальних змінних або замінювати ресурсоємні бібліотеки. Такий моніторинг дозволяє уникнути збоїв і забезпечити стабільність роботи прошивки.

3.2 Прототип на платформі Лілка

Перехід від платформи Arduino до використання контролера Лілка (на базі ESP32) був зумовлений як технічними, так і концептуальними

міркуваннями. На етапі побудови базового прототипу Arduino продемонструвала свою ефективність у швидкій реалізації логіки сенсорного зчитування та інтерфейсної взаємодії. Однак обмеження апаратної платформи – передусім дефіцит оперативної пам’яті, відсутність апаратної підтримки Wi-Fi, а також обмежена кількість периферійних інтерфейсів – стали значущими факторами для переходу до більш потужного рішення.

Консоль Лілка, побудована на ESP32, забезпечує суттєво більші можливості: збільшений обсяг оперативної пам’яті (до 512 КБ SRAM), наявність вбудованого Wi-Fi та Bluetooth, багатоканальні АЦП, підтримку FreeRTOS. Усе це дозволяє перейти від лінійної логіки опитування сенсорів до побудови асинхронної, подієво-орієнтованої архітектури з можливістю розширення. З практичної точки зору, перехід до Лілки відкрив можливість реалізації наступних функцій:

- логування інформації із записом на карту пам’яті;
- синхронізація даних по Wi-Fi з віддаленим сервером;
- інтеграція web-інтерфейсу для керування параметрами системи;
- використання більш важких бібліотек і алгоритмів обробки без ризику вичерпання пам’яті.

Arduino-прототип був зосереджений на одномоментному зборі експозиційних параметрів, а Лілка дозволяє реалізувати повноцінну екосистему збору, передачі, аналізу й візуалізації даних. У подальшому це відкриває шлях до інтеграції алгоритмів машинного навчання, віддаленого оновлення прошивки, а також створення адаптивної поведінки пристрою залежно від умов середовища.

3.2.1 Загальний алгоритм

Загальний алгоритм роботи системи (рисунок 3.3) після переходу на платформу Лілка (ESP32) суттєво відрізняється від попередньої реалізації на Arduino. Основна зміна полягає не лише в оновленні апаратної бази, а й у

зміні структурної логіки: система наближається до повноцінної модульної архітектури з чітким розділенням функціональних компонентів і відповідальністю кожного модуля.

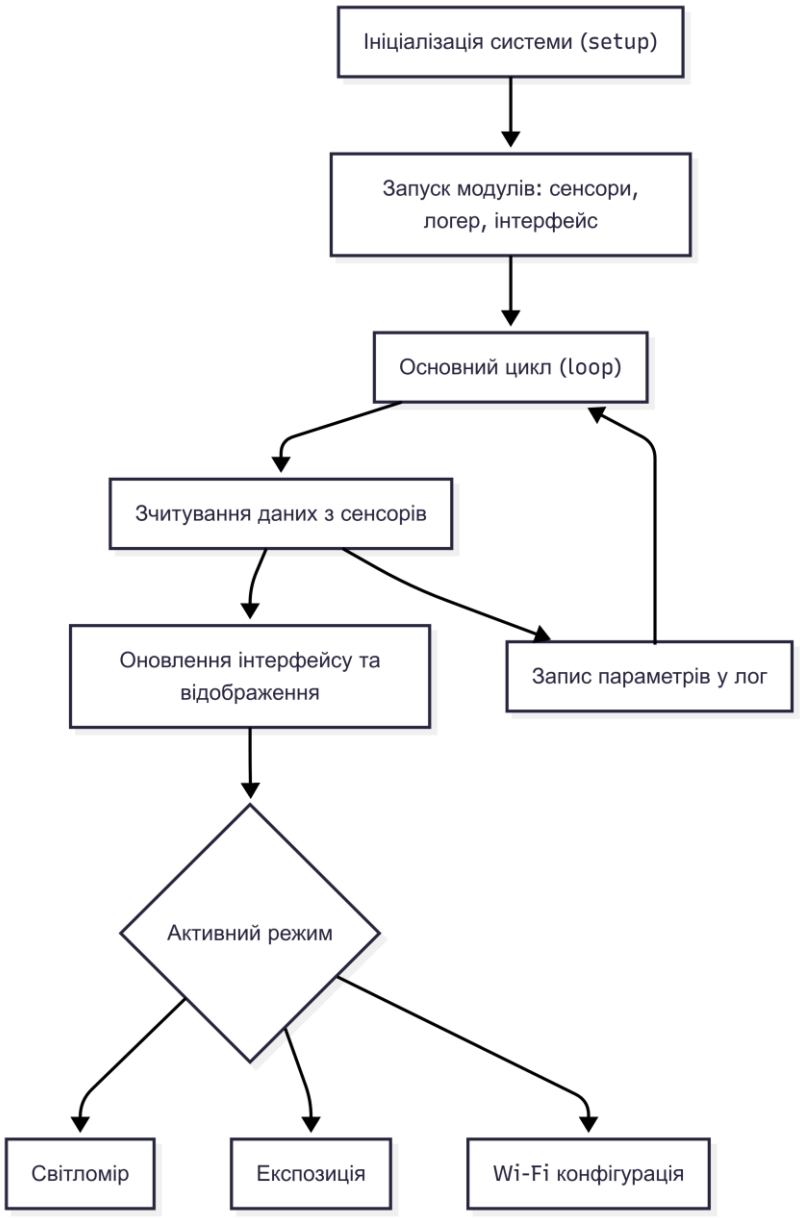

Рисунок 3.3 – Загальний алгоритм роботи прототипу на Лілці

Початкова фаза – setup() – виконує стандартну ініціалізацію всіх модулів. Вона включає запуск контролера Lilka, конфігурацію I2C-шини (із явно заданими пін-контактами SDA/SCL), запуск головного меню, сенсора освітлення та модуля логування. Кожен із цих компонентів реалізований як окремий об’єкт із власним методом begin(), що дозволяє централізовано та

ізольовано виконувати ініціалізацію. Такий підхід відповідає сучасним принципам компонентно-орієнтованого дизайну у вбудованих системах.

Основний цикл loop() реалізує опитування датчиків, обробку логіки інтерфейсу та оновлення візуального відображення. Спочатку створюється об’єкт типу lilka::Canvas, який є буфером рендерингу для відображення графічних компонентів. Після цього отримується поточний стан системи (lilka::controller.getState()), що використовується для реактивної обробки подій у відповідних модулях інтерфейсу. Зчитування сенсорних даних відбувається безпосередньо через об’єкт LightSensor, який повертає значення освітленості (у люксах), колірної температури (CCT) та експозиційної величини (EV). Ці значення передаються у відповідні модулі:

- об’єкт Exposure оперує встановленням параметрів ISO та апертури, розраховує витримку;

- об’єкт Logger виконує логування параметрів разом із системним станом.

Основною особливістю реалізації на Лілці є реактивне перемикання між інтерфейсними режимами залежно від стану меню. Умовні блоки перевіряють, який розділ інтерфейсу наразі активний, і відповідно викликають методи для його обробки та візуалізації. Якщо меню не вибрано – малюється список розділів. Якщо обрано режим світломіра – відображаються показники освітлення та температури світла.

У режимі експозиції користувач може змінювати параметри зйомки, після чого інтерфейс візуалізує поточний EV, рекомендовані межі EV (відповідно до типових витримок, таких як 1/60 або 1/500) та розраховану витримку. Режим конфігурації Wi-Fi дозволяє працювати з параметрами з’єднання та передачі даних у логгері.

Поточна реалізація loop() має певні риси процедурного підходу, притаманного Arduino-сумісному коду, зокрема послідовне виконання всіх дій у циклі без використання асинхронного планування або багатопоточності. Застосування delay() тут навмисно включено, і відсутність event- або task-

driven підсистем поки свідчить про те, що ця логіка є проміжним етапом на шляху до повноцінного асинхронного рішення, яке можливе завдяки FreeRTOS, доступному в ESP32.

Прототип дозволяє перевірити цілісну інтеграцію основних компонентів, оцінити реактивність інтерфейсу, точність зчитування сенсорних даних та стабільність логування. Після цього доцільно перейти до оптимізації: винесення сенсорного опитування в окремі task-и, реалізація подій і підсистем обробки повідомлень та перехід до енергозберігаючих режимів.

3.2.2 Модуль LightMeter

Модуль LightMeter (додаток Б.2.1) є візуальним компонентом системи логування світлового середовища на базі Lilka-платформи. Його основна функція полягає в наочному відображенні параметрів навколишнього освітлення, зокрема, освітленості (у люксах) та корельованої колірної температури (CCT, у Кельвінах) у вигляді інтуїтивно зрозумілого графічного інтерфейсу. Це не лише полегшує сприйняття параметрів, а й дозволяє проводити швидкий аналіз умов зйомки або вимірювання візуально, без потреби у складному числовому аналізі. Модуль успадковується від базового класу MenuComponent, що надає базову логіку для інтеграції з інтерфейсом меню.

Метод drawUI() є основним публічним інтерфейсом модуля, через який відбувається побудова візуального інтерфейсу. У цьому методі відбувається послідовний виклик функцій виводу графічних елементів: шкали освітленості (Lux bar), шкали колірної температури (Kelvin bar), а також перехресного візуального індикатора – так званого crosshair – який показує поточне значення освітленості та CCT на відповідних шкалах. Таким чином, модуль працює у режимі зчитування двох параметрів та їх паралельного виведення на екран, з можливістю точного візуального позиціонування.

Побудова шкали освітленості (lux bar) реалізується у методі drawGradientLuxBar(). Цей елемент розміщується вертикально і заповнюється градієнтом від білого до чорного кольору. Для кожного горизонтального рядка (пікселя) у межах висоти шкали обчислюється відповідний відтінок сірого кольору, який потім перетворюється у 16-бітний формат і виводиться прямокутником висотою в 1 піксель.

Таким чином створюється плавна градація, яка дозволяє візуально оцінити інтенсивність освітлення, причому найбільш яскраві ділянки (верх шкали) відповідають високим рівням освітлення, а найтемніші (низ) – низьким.

Шкала колірної температури (Kelvin bar), яку будує метод drawGradientKelvinBar(), реалізована горизонтально. На відміну від шкали освітленості, ця шкала потребує складнішої логіки побудови, оскільки включає кольоровий градієнт, що відповідає загальноприйнятим візуальним уявленням про температуру світла: від теплого червоно-жовтого (1800–3000K), через нейтральне біле (5000K), до холодного блакитного (6600K). Для побудови цієї шкали використовується масив кольорових стопів (color stops), кожен з яких має задану температуру та відповідне значення в RGB. Кожен піксель шкали інтерполюється між найближчими ключовими точками, що дозволяє отримати плавний перехід між кольорами. Цей підхід з інтерполяцією забезпечує високий ступінь відповідності візуальної шкали до реальних умов освітлення.

Метод drawCrosshair() є центральною частиною інтерфейсу, оскільки відповідає за точне візуальне відображення поточних параметрів освітлення. Він будує дві лінії: горизонтальну (від шкали люксів до точки перетину) та вертикальну (від шкали Кельвінів). У місці перетину цих ліній розміщується заповнене коло зеленого кольору, що виконує функцію маркера поточного стану. Біля цієї точки відображаються числові значення lux і Kelvin. Для кращої візуальної якості застосовується центрування тексту та адаптивне розміщення відносно точки перетину.

Додаткові функції mapLuxToY() та mapKelvinToX() відповідають за трансформацію фізичних значень lux та Kelvin у координати піксельного простору екрана. У випадку lux використовується масив опорних значень, який відображає логарифмічну природу людського сприйняття яскравості. Значення lux нормалізується відносно цього масиву, і обчислюється проміжна позиція між найближчими стопами. Це дозволяє точно масштабувати діапазон значень на вертикальній шкалі. У випадку колірної температури використовується лінійне масштабування між мінімальним (1800K) і максимальним (6600K) значенням по горизонталі. Така логіка дозволяє використовувати повну ширину шкали без артефактів або викривлень.

У процесі виводу використовуються шрифти FONT_6x12 і setTextSize(2), що дозволяє забезпечити читання значень на обмеженому екрані.

Колір тексту – білий, що створює високий контраст із кольоровими та градієнтними фонами. Виведення значень lux і Kelvin відбувається тільки після побудови шкал і ліній, що дозволяє уникнути накладання елементів і візуального шуму.

З архітектурної точки зору модуль є прикладом успішного інкапсульованого підходу: весь вивід, трансформації координат, логіка шкал та інтерфейсної взаємодії зосереджені в межах одного класу. Це спрощує супровід коду та його повторне використання в інших компонентах або проектах. Крім того, модуль не має зовнішніх залежностей від логіки контролера або системного стану, працюючи виключно на підставі переданих значень параметрів освітлення.

Загалом, LightMeter виконує не лише функцію відображення поточних параметрів, а й є важливим інструментом для користувача: він дозволяє швидко оцінити світлове середовище, прийняти рішення щодо налаштувань експозиції та зробити висновки про зміну умов у динаміці. Такий візуальний інтерфейс особливо корисний у контексті аналогової або експериментальної

фотографії, де пряма оцінка параметрів освітлення критично важлива. Модуль реалізує одну з складових інтерфейсу в системі логування світлового середовища, об’єднуючі дані апаратного зчитування і візуальну репрезентацію в єдину, логічно цілісну одиницю.

3.2.3 Модуль Exposure

Модуль Exposure (додаток Б.2.2) у складі системи логування світлового середовища забезпечує інтерфейс користувача для встановлення та візуалізації основних параметрів експозиції, а саме чутливості (ISO), діафрагми (aperture) та обчисленої витримки (shutter speed). Його реалізація базується на спадкуванні від абстрактного класу MenuComponent, що дозволяє використовувати загальні методи інтерфейсної взаємодії, зокрема малювання спільних елементів UI.

Основною внутрішньою структурою модуля є перелік фіксованих дискретних значень параметрів ISO та діафрагми. Масив ISO_VALUES містить 16 елементів, починаючи з умовного нульового значення та закінчуючи ISO 6400. Аналогічно, масив APERTURE_VALUES охоплює 12 рівнів діафрагми – від f/1.4 до f/64, також із початковим нульовим елементом. Кожен з цих масивів використовується індексовано: інтерфейс зберігає поточний вибраний індекс і змінює його за допомогою логіки обробки подій.

Перелік (enum) FL_Parameter вказує на поточний стан інтерфейсу, тобто який саме параметр обраний для редагування – ISO, діафрагма або жоден (NONE). Перемикання між параметрами здійснюється за допомогою обробки стану кнопок, отриманих через об’єкт типу lilka::State. Кнопки UP та DOWN перемикають вибір між параметрами, а LEFT/RIGHT – циклічно змінюють значення поточного параметра.

Функція drawUI відповідає за побудову візуального відображення всіх трьох параметрів: ISO, діафрагми та витримки. Вона також виводить шкалу експозиційних значень EV з числовими позначками та індикаторами.

Параметри ISO та aperture відображаються з квадратними дужками, наприклад: ISO [100], F/[2.8]. Поточний вибір (виділення) параметра супроводжується зміною кольору тексту на жовтий, що покращує сприйняття. Значення витримки (shutter) обчислюється з урахуванням EV, ISO та діафрагми через зовнішній модуль (LightSensor) і подається як обернене значення у вигляді 1/X.

Окрема увага приділена візуалізації відповідності обраних параметрів рекомендованим межам EV. Колір витримки змінюється відповідно до оцінки:

- зелений (evDifferenceOK) – EV потрапляє в рекомендований інтервал;
- помаранчевий (evDifferenceWARN) – EV перебуває на межі;
- червоний (evDifferenceCRIT) – EV виходить за межі допустимого.

Це дозволяє користувачу оперативно оцінити якість поточної експозиції без необхідності числового аналізу. У нижній частині інтерфейсу виводиться горизонтальна шкала значень EV. Вона складається з центральної осі та вертикальних рисок через кожні 5 одиниць, з числовими підписами. Колірна індикація також присутня:

- поточне значення EV відображається червоною вертикальною рискою;
- рекомендовані межі (мінімальна та максимальна EV) позначено зеленими лініями.

Над кожною лінією додатково виводиться точне числове значення EV з одним десятковим знаком. Це створює наочне уявлення про положення поточної експозиції щодо меж оптимального діапазону.

Обробка взаємодії з користувачем реалізована через метод handleParameters, який приймає структуру lilka::State – інкапсульоване представлення стану натискань кнопок. Логіка дозволяє циклічно перемикати вибраний параметр (ISO – APERTURE – NONE) та змінювати значення параметра в межах відповідного масиву із циклічним переходом.

Зміна значень забезпечується допоміжним методом getNextIndex, який гарантує обробку переповнень і повернень до початку списку. Таким чином, інтерфейс не обмежує користувача при навігації та не допускає помилкових станів.

Для зручності реалізовано три функції-оцінювачі, які використовуються при відображенні кольору витримки, як описано вище:

- evDifferenceOK – повертає true, якщо значення EV строго потрапляє між мінімумом і максимумом;
- evDifferenceWARN – охоплює випадки, коли значення дорівнює одній з меж;
- evDifferenceCRIT – покриває ситуації, коли значення виходить за встановлені межі.

Модуль Exposure реалізує повноцінну систему ручного налаштування параметрів експозиції з візуальною підтримкою та адаптивною підказкою. Його архітектура дозволяє як ефективно взаємодіяти з користувачем, так і забезпечувати гнучке розширення в майбутньому (наприклад, додавання нового параметра або нових режимів обробки EV). Чітке розмежування між даними, логікою відображення та обробкою подій дозволяє підтримувати модуль, а структура з фіксованими масивами значень забезпечіє стабільність роботи на ресурсно-обмежених пристроях, таких як Lilka або Arduino-сумісні платформи.

3.2.4 Допоміжні класи

У системі допоміжні класи FL_Menu (додаток Б.2.3) та MenuComponent (додаток Б.2.4) виконують роль базових будівельних блоків користувацького інтерфейсу. Вони забезпечують навігацію між модулями та стандартизоване оформлення інтерфейсів, відповідно. Використання класів FL_Menu та MenuComponent значно спрощує розширення системи, додаючи структурованість та узгодженість UI-компонентів.

FL_Menu – це спеціалізований головний екран системи, спадкоємець класу lilka::Menu, який надає базову реалізацію меню з переміщенням по пунктам. У конструкторі FL_Menu() відбувається ініціалізація з заголовком меню. Цей рядок відображається як шапка головного меню на дисплеї. Метод begin() виконує початкову ініціалізацію, додає до меню три основні пункти:

- світлометрія – відображення поточних значень освітлення (lux) і температури кольору (CCT);
- експозиція – ручне налаштування параметрів з графічною шкалою EV;
- мережа – інтерфейс для Wi-Fi конфігурації та синхронізації з сервером.

Патерн currentMenuInstance дозволяє статичній функції staticSetSelected отримати доступ до поточного екземпляру меню. Це рішення потрібне для того, щоб передати виклик натискання кнопки START у вигляді функції-обробника (callback), прийнятого контролером. Саме ця функція дозволяє повертатися з підменю назад у головне меню.

Метод drawMenu() відповідає за запуск основного циклу взаємодії користувача з меню. Він оновлює стан і перемальовує екран, поки користувач не вибере пункт (що викличе метод isFinished()).

Методи isLightMeter(), isExposure() та isWiFiConfig() інкапсулюють логіку перевірки, який саме пункт було обрано. Вони порівнюють індекс курсора з елементами enum FL_Menu_Enum, дозволяючи головному циклу програми визначити, який модуль відобразити в поточний момент.

MenuComponent – це базовий клас для всіх UI-компонентів (як Exposure, LightMeter тощо). Його завдання – надати спільні функції, які мають бути присутні в кожному екранному інтерфейсі.

На даному етапі цей клас має лише один метод – drawCommonUI(). Він виконує базове очищення екрана та вивід текстової підказки в нижній частині екрана. Це дозволяє користувачу зрозуміти, як повернутися в головне меню. Такий підхід забезпечує єдиний UX-патерн у всіх екранах.

Допоміжні класи створюють каркас для розгортання інтерфейсу, з чіткою відокремленістю навігаційної логіки від візуального рендерингу. У майбутньому це дозволить додавати нові екрани та компоненти без дублікації коду або втрати консистентності в поведінці системи.

3.3 Робота з даними

У системі реалізовано два способи збереження логів – локально на картці пам’яті microSD та віддалено у базі даних PostgreSQL. Така стратегія подвійного збереження дозволяє забезпечити як незалежність від доступу до мережі, так і надійність у випадку втрати одного з каналів. На етапі польових вимірювань дані завжди спочатку потрапляють на карту пам’яті, що гарантує їх фіксацію незалежно від якості Wi-Fi-з’єднання або наявності інтернету загалом.

Запис на microSD реалізується у вигляді структурованих CSV файлів, що дозволяє легко обробити ці дані офлайн або імпортувати їх у сторонні інструменти. Такий підхід особливо зручний для резервного копіювання або ретроспективного аналізу результатів світлометрії та експозиції на знімальному майданчику.

Одночасно, при наявності стабільного Wi-Fi-з’єднання, дані передаються до бази PostgreSQL. Це забезпечує централізоване зберігання та легкий доступ до історії з вебінтерфейсу. Структура бази передбачає зберігання параметрів освітлення, експозиційних значень, температури світла та налаштування сенсорів вимірювання, що дозволяє здійснювати подальшу статистичну обробку. Використання двох засобів збереження (локального та віддаленого) відкриває можливості для гібридної роботи: наприклад, автоматичної синхронізації логів з SD-карти у базу після відновлення мережі. Така архітектура дозволяє системі залишатися незалежною, адаптивною до середовища та надійною в контексті професійного використання в умовах обмеженої інфраструктури.

3.3.1 Модуль Logger

Модуль Logger (додаток Б.2.5) у системі відповідає за локальне збереження виміряних даних на карті пам’яті у вигляді CSV файлу. Він побудований з урахуванням розширюваності, інтеграції з Wi-Fi та синхронізації з віддаленим сервером. Модуль Logger забезпечує надійний спосіб логування експозиційних параметрів та характеристик освітлення, таких як ISO, діафрагма, витримка, освітленість (lux), температура світла (CCT) та значення експозиції (EV). Для кожного запису також зберігається унікальний ідентифікатор (UUID) та маркер синхронізації.

Клас реалізує три режими роботи: SUSPENDED (жодних дій не виконується), SINGLE (разовий запис по натисканню кнопки) та STREAM (потокове логування через певний інтервал). Перемикання між режимами відбувається за допомогою кнопок, що обробляються в методі handleLogging. У режимі STREAM дані зберігаються постійно з заданим інтервалом, тоді як у SINGLE логування відбувається лише раз після натискання відповідної кнопки. Це дозволяє адаптувати роботу пристрою до різних сценаріїв використання: польове дослідження, тестування або фотозйомка.

Метод saveData відкриває файл /fl_logs.csv у режимі додавання та зберігає новий рядок даних. Кожен рядок містить значення, розділені комами, включно з полем synced, яке дорівнює 0 за замовчуванням. Це поле існує для маркування, чи був запис вже відправлений у віддалену базу даних. Метод generateUUIDv4 генерує унікальний ідентифікатор згідно зі стандартом UUID v4 на основі випадкових байтів з апаратного генератора ESP32, що дозволяє точно ідентифікувати кожен запис навіть після експорту.

Для читання даних, які ще не були синхронізовані, реалізовано метод readRecords, що повертає JSON масив з об’єктами, кожен з яких містить поля, відповідні колонкам CSV файлу. Цей метод буде використаний для підготовки масиву до надсилання на сервер. При цьому синхронізовані записи (з позначкою synced = 1) ігноруються. Щоб помітити вже оброблені

рядки як синхронізовані, існує метод markAsSynced, який переписує файл, змінюючи значення останньої колонки з 0 на 1 для певної кількості записів (обмежено аргументом limit). Це реалізовано через створення тимчасового файлу з оновленим вмістом, після чого старий файл замінюється.

Метод countUnsyncedRecords дозволяє підрахувати кількість рядків, які ще не були синхронізовані з сервером. Це корисно як для діагностики, так і для планування пакетної відправки даних.

Додатково метод applySettings дозволяє налаштувати інтервал потокового логування через JSON об'єкт конфігурації, отриманий з мережевого інтерфейсу.

Logger дозволяє надійно фіксувати та маркувати параметри освітлення. Завдяки гнучкій підтримці трьох режимів роботи, можливості інтеграції з Wi-Fi логікою та структурованому формату даних, цей модуль виступає центральною точкою збирання інформації для подальшої аналітики та синхронізації з зовнішніми базами, такими як PostgreSQL. Його реалізація враховує обмеження платформи ESP32, зокрема малий обсяг оперативної пам’яті, тому зберігання та обробка даних відбувається поетапно, без потреби завантаження всього вмісту у RAM.

3.3.2 Мережева взаємодія

Модулі NetworkService (додаток Б.2.6) та APIService (додаток Б.2.7) забезпечують взаємодію пристрою з віддаленим сервером через HTTP запити. Вони інкапсулюють логіку обміну даними, виконання запитів до API та обробку відповідей. Архітектурно NetworkService виконує роль низькорівневого HTTP-клієнта, тоді як APIService працює як високорівневий фасад, що забезпечує більш абстрагований доступ до конкретних ендпоінтів API. Клас NetworkService призначений для здійснення HTTP запитів (GET, POST, PATCH) до зовнішнього сервера API. Він використовує бібліотеку HTTPClient для створення з’єднань та відправки даних. Кожен метод

приймає suburl – частину адреси після базового URL, визначеного у файлі .secrets.h. Цей підхід дозволяє легко змінювати адреси без модифікації системного коду.

У методі getURL формується повна адреса, додаючи suburl до базової константи API_URL. Метод jsonify відповідає за перетворення рядка-відповіді від сервера у формат JsonDocument за допомогою ArduinoJson. Якщо запит завершився з помилкою або відповідь порожня, jsonify генерує спеціальний JSON об’єкт із полем message та порожнім полем data. Для кодування рядків у формат URL реалізовано метод encodeURL. Він перетворює спеціальні символи (наприклад, пробіли або кирилицю) у шістнадцяткове представлення, згідно зі стандартом URL-encoding. Це необхідно для безпечної передачі даних, наприклад, у GET запитах.

Усі запити перевіряють, чи встановлене з’єднання з Wi-Fi. У разі відсутності підключення функція повертає об’єкт-помилку. Методи get, post і patch обробляють початок, виконання й завершення HTTP з’єднання. У POST та PATCH додаються специфічні заголовки (content-type) і виконується серіалізація JsonDocument у текстовий формат. У випадку успішного запиту (200 або 201 код) відповідь парситься і повертається у вигляді JsonDocument.

Методи getMessage та getData спрощують доступ до структури відповіді сервера, зокрема отримання повідомлення (message) або вкладених даних (data). Такий поділ дозволяє ізолювати логіку обробки HTTP та JSON від прикладного рівня.

Клас APIService є обгорткою над NetworkService, яка надає функції для взаємодії з конкретними ендпоінтами REST API. Він реалізує такі методи: checkConnection, exportRecords, getTSC34725Settings, getLoggingSettings. Кожен із них викликає відповідну функцію з NetworkService, обробляє результат та повертає зручну для обробки в інших модулях відповідь.

Метод checkConnection використовується для перевірки з’єднання з сервером. Він надсилає GET запит на ендпоінт /api/v1/system, передаючи тестове повідомлення, попередньо закодоване через encodeURL. Результат

обробляється функцією handleStringApiResponse, яка друкує повідомлення сервера в консоль і повертає текстову відповідь (якщо об’єкт data не порожній).

Метод exportRecords відповідає за відправлення локально збережених записів (логів освітлення) на сервер. Він приймає JsonDocument, що містить масив даних, який пакується у поле data нового JSON об’єкта, після чого передається на ендпоінт /api/v1/light-records/import через POST запит. У разі успіху сервер повертає підтвердження, яке виводиться в консоль.

Метод getTSC34725Settings дозволяє отримати конфігурацію для сенсора освітлення (TCS34725). Після виконання GET запиту на /api/v1/sensors, з поля data вилучається масив rows. Якщо масив не порожній, з першого об’єкта витягуються параметри сенсора: type, gain, integrationTime. Ці параметри записуються у новий JsonDocument, який повертається. Така реалізація дає змогу зберігати централізовані налаштування для сенсорів на сервері, синхронізуючи їх з пристроєм.

Метод getLoggingSettings працює подібно до попереднього, але звертається до ендпоінту /api/v1/logging. Його мета – отримати конфігураційні параметри логування, наприклад, інтервал потокового режиму. Після виконання запиту поле data із відповіді повертається напряму для подальшої обробки.

Важливою особливістю цієї архітектури є чітке розділення обов’язків: NetworkService лише виконує мережеві запити, тоді як APIService пов’язаний з елементами доменної області: sensors, logging, records тощо. Це відповідає класичній моделі Service Layer, яка полегшує підтримку та тестування коду, а також спрощує розширення – достатньо додати новий метод до APIService, не змінюючи внутрішню логіку передачі даних.

Обробка відповідей виконана з урахуванням обробки помилок та пустих відповідей. Завдяки JSON обгортці кожен виклик API має чітко визначену структуру: поле message для повідомлення користувача, data – для payload, а rows у масивах для табличних результатів. Цей підхід уніфікує

роботу з API незалежно від типу ендпоінту. Архітектура NetworkService з APIService легко масштабується. Наприклад, можна додати PUT, DELETE або підтримку WebSocket без порушення наявного інтерфейсу. Крім того, у разі переходу на інший транспорт (наприклад, MQTT) достатньо реалізувати нову низькорівневу службу, зберігши структуру APIService.

3.3.3 Модуль WiFiConfig

Модуль WiFiConfig (додаток Б.2.8) відповідає за підключення пристрою Lilka до Wi-Fi мережі, ініціалізацію з'єднання з сервером, синхронізацію локальних даних та оновлення апаратних налаштувань пристрою через API. Він побудований як графічне меню на базі MenuComponent і включає UI з прогресбаром, логіку повторних підключень і логування. Вся логіка взаємодії побудована на використанні сервісів APIService, Logger та LightSensor, що забезпечує повну інтеграцію з екосистемою. Метод begin запускає процес підключення до мережі, використовуючи заздалегідь визначені змінні WIFI_SSID і WIFI_PASSWORD, що зберігаються у файлі .secrets.h. Підключення не перевіряється одразу – вся перевірка й обробка відбувається в методі drawUI, який поєднує в собі логіку підключення, обробку стану та UI.

Графічна частина реалізована за допомогою lilka::ProgressDialog, який показує статус підключення до мережі, використовуючи простий прогресбар. У циклі while виконується maxAttempts (100 за замовчуванням) перевірок статусу підключення. На кожному кроці оновлюється прогресбар, викликається begin, виконується коротка пауза, і повторюється спроба. Це дозволяє інформувати користувача про хід процесу та уникнути зависань інтерфейсу. У разі успішного з'єднання, викликається initConnection або syncData залежно від стану connectionChecked. Результат (текстове повідомлення) записується у progress.setMessage(). В іншому випадку, якщо Wi-Fi недоступний, виводиться повідомлення про помилку.

Метод initConnection використовується для перевірки зв’язку з сервером та оновлення конфігурації пристрою. У ньому створюється об'єкт APIService, через який викликається checkConnection(). Якщо відповідь не порожня (що вказує на успішне з’єднання), виконується updateHardware, що намагається витягнути з сервера налаштування сенсора (TSC34725) і логера.

Метод updateHardware створює об'єкти APIService та LightSensor, після чого запитує серверні налаштування для сенсора і логування. Дані застосовуються відповідними методами applySettings. Метод повертає відповідне повідомлення про успіх або помилку, що дозволяє пристрою працювати із централізовано керованими параметрами, не потребуючи перекомпіляції прошивки.

Важливою частиною логіки є параметр connectionChecked, який гарантує, що оновлення налаштувань виконується лише один раз після з’єднання. Це дозволяє уникнути повторного запиту налаштувань, коли користувач заходить у це меню повторно в рамках одного сеансу.

Метод syncData – це функція для відправлення локально збережених логів на сервер. Він створює об’єкти APIService та Logger, після чого викликає readRecords(limit) для зчитування (до 10) рядків логів, які ще не були відмічені як синхронізовані. Якщо є записи, дані відправляються на сервер методом exportRecords.

Якщо відправлення пройшло успішно, викликається markAsSynced, який помічає щойно відправлені рядки як синхронізовані, оновлюючи відповідне поле в CSV файлі на SD-карті. Потім викликається countUnsyncedRecords для визначення, скільки рядків ще залишились. Це дає змогу показати користувачу, скільки даних уже передано, а скільки ще залишилося. У разі помилки на будь-якому етапі, метод повертає відповідне текстове повідомлення, яке виводиться на екран. У випадку, якщо нових даних немає, користувач отримує відповідне повідомлення. Таким чином, користувач завжди розуміє, що саме відбувається: чи передані дані, чи виникла помилка, чи просто нічого не було для синхронізації.

Інтерфейс оновлюється кожного разу через progress.draw(canvas) та lilka::display.drawCanvas(canvas), що забезпечує користувачеві актуальну інформацію. Наприкінці виконується delay(3000), щоб користувач встиг прочитати повідомлення – але у майбутньому це бажано замінити на millis, щоб уникнути блокувань.

3.4 Серверна частина

Як віддалене сховище даних використовується база даних у PostgreSQL, модель якої описана за допомогою Prisma ORM (додаток Б.3.1). Цей типобезпечний інструмент роботи з базами даних дозволяє генерувати зручний клієнт на основі декларативної схеми. У системі зберігаються три основні типи даних: світлові записи (LightRecord), конфігурація сенсора (Sensor) та налаштування логування (Logging). Усі таблиці мають поля для аудиту (createdAt, updatedAt) та використовують UUIDv4 як первинний ключ.

3.4.1 Моделі в базі даних

Модель LightRecord (додаток Б.3.2) є центральною таблицею для збереження логів, що надходять з пристрою. Вона містить дані про параметри експозиції та освітлення: ISO, діафрагму (aperture), витримку (shutter), освітленість (lux), колірну температуру (cct) і експозиційне значення (ev). Поле shutter є nullable, що дозволяє обробляти випадки, коли інформація про витримку відсутня (наприклад, при тестових вимірюваннях). Усі інші параметри є обов’язковими. Дані вносяться у цю таблицю через API, який отримує JSON з пристрою.

Модель Sensor (додаток Б.3.3) призначена для централізованого збереження поточних налаштувань світлочутливого сенсора, зокрема, коефіцієнта підсилення (gain) і часу інтеграції (integrationTime). Важливо, що поле type є унікальним, тобто для кожного типу сенсора в базі може

зберігатися лише один запис. Це дозволяє уникнути дублювання і забезпечує консистентність: пристрій запитує налаштування один раз і точно отримує саме те, що потрібно. Підтримка типу сенсора дозволяє масштабувати систему: у майбутньому можна буде додати підтримку інших сенсорів, просто розширивши enum SensorType.

Модель Logging (додаток Б.3.4) зберігає конфігурацію логера, який визначає інтервал автоматичного логування в секундах (streamIntervalSec). Ця таблиця дозволяє віддалено управляти частотою запису даних, не змінюючи прошивку пристрою.

Такий підхід відкриває можливості для адаптивної оптимізації – наприклад, підвищити частоту при високій динаміці освітлення, або зменшити її для збереження енергії в стабільних умовах. Пристрій запитує це значення через API і оновлює свій локальний конфіг.

Для ініціалізації системи використовується скрипт (додаток Б.3.5), який створює стандартні записи в таблицях Sensor та Logging (рисунок 3.4). Це гарантує, що при першому запуску сервер вже буде мати валідні налаштування, з якими може працювати пристрій. Наприклад, створюється запис для сенсора типу TCS34725 із типовими визначеними параметрами (підсилення та інтеграція), а також конфіг логера з інтервалом 1 секунда.

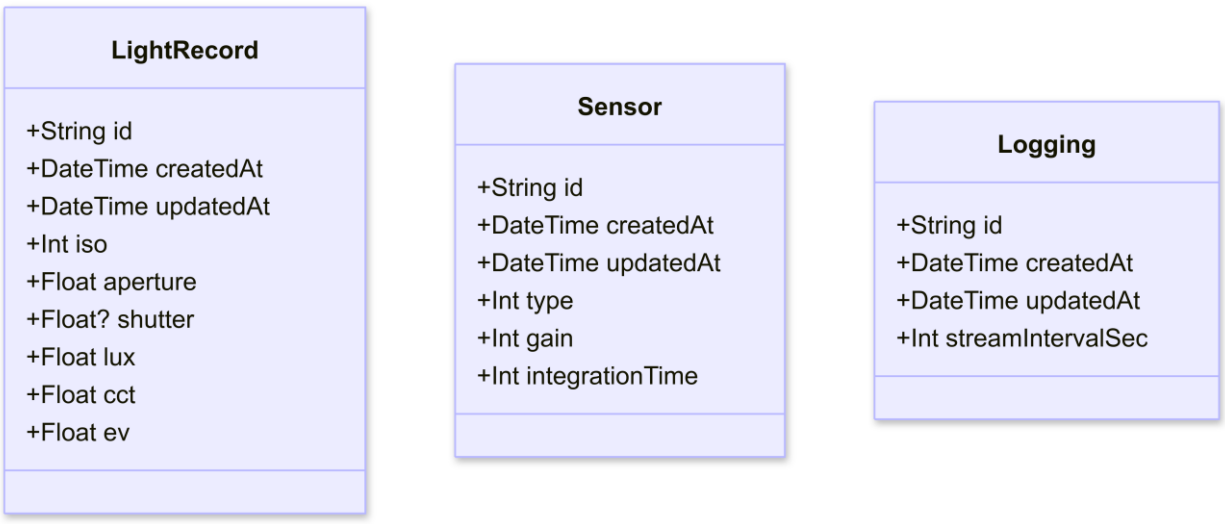

Рисунок 3.4 – UML представлення моделей в базі даних

Усі схеми моделей (рисунок 3.4) побудовані з урахуванням майбутнього масштабування. Наприклад, поля createdAt і updatedAt дозволяють легко відстежувати час останньої модифікації, що важливо як для внутрішньої аналітики, так і для відлагодження. Використання UUID як ключа гарантує унікальність записів незалежно від джерела, що дозволяє одночасну роботу з кількома пристроями без ризику конфліктів.

Загалом, прозорість та розширюваність бази даних дозволяє реалізувати повноцінний цикл: вимірювання – логування – збереження – аналітика.

3.4.2 Архітектура бекенду

Код серверної частині реалізовано у чітко структуровану багаторівневу архітектуру, засновану на принципах розділення відповідальності (Separation of Concerns). Основними рівнями є: конфігураційний, маршрутизації, контролерів, сервісів, маперів, хендлерів (обробників), схем валідації та інфраструктурних абстракцій. Такий підхід забезпечує масштабованість, тестованість і зручність підтримки коду в довготривалій перспективі.

На базовому рівні знаходиться конфігураційний модуль, який відповідає за ініціалізацію параметрів оточення, таких як порт, ліміти JSON, секретні ключі, URL до бази даних (додаток Б.3.6).

Це дозволяє адаптувати систему до різних середовищ розгортання, наприклад для локальної розробки. Конфігурація централізована, що спрощує контроль за залежностями та запобігає розосередження логіки ініціалізації по проекту.

Наступним рівнем є рівень запуску застосунку (вхідна точка app.ts, наведена на додатку Б.3.7). Тут відбувається підключення middleware (обробка CORS, парсинг JSON), а також реєстрація основних маршрутів API, згрупованих у версію (/api/v1). Таким чином, можна підтримувати декілька версій API паралельно, не зачіпаючи стабільну роботу вже існуючих клієнтів.

Роутери є кінцевими точками (endpoints) до логіки кожного модуля. Вони делегують запити на відповідні контролери, а також застосовують проміжні обробники (middleware) для валідації даних. Роутери згруповані за доменами: логування (logging), світлові записи (light-records), сенсори (sensors) та системні (system). Такий підхід чітко розділяє відповідальність між різними модулями системи.

Контролери реалізують логіку обробки запитів, приймаючи дані, що вже пройшли валідацію, викликаючи відповідні сервіси, та повертаючи уніфіковані відповіді. Вони не взаємодіють безпосередньо з базою даних, а делегують цю відповідальність сервісам. Це дозволяє уникнути дублювання логіки, зокрема перевірок, логування та формування результатів.

Крім того, у контролерах застосовується декоратор logger, який автоматично виводить у консоль інформацію про запити та відповіді, а також фіксує помилки.

Сервіси виступають ядром бізнес-логіки. Вони безпосередньо працюють з клієнтом бази даних Prisma, реалізуючи CRUD операції, обробку логіки пагінації тощо. Усі сервіси наслідують абстрактний клас SimpleService (додаток Б.3.8), який містить допоміжні методи для обчислення offset та limit.

Мапери виконують роль трансформаторів між моделями з бази даних і DTO (data transfer object), що відправляються клієнту. Вони наслідують базовий клас SimpleMapper (додаток Б.3.9), де вказуються поля, які потрібно брати з моделей, та перетворюють об’єкти у зовнішньо безпечний формат. Завдяки цьому рівню до клієнта не потрапляють зайві або чутливі дані.

Обробники запитів (handlers) відповідають за різну логіку, наприклад, окремо виділено класи для роботи з валідацією параметрів та CSV (додаток Б.3.10), а також для формування уніфікованої відповіді (додаток Б.3.11). Ці обробники допомагають централізувати спільні патерни: усі відповіді клієнту йдуть у єдиному форматі – з полем message та об’єктом data, що полегшує інтеграцію з фронтендом або пристроями. Також через окремі хендлери реалізовано фіксацію часу запиту і відповіді.

Ще одним рівнем архітектури є система валідації даних. Вона побудована на Joi – бібліотеці для валідації структури об’єктів. Для кожного типу запиту описано схеми, які перевіряють формат, обов’язковість та діапазони даних. Це дозволяє не лише захистити сервер від некоректних або шкідливих запитів, але й зробити розробку більш передбачуваною. Наприклад, ще до потрапляння в бізнес-логіку перевіряється структура масиву записів або наявність правильного UUID.

Вся архітектура підтримує принцип інверсії залежностей: фабрики створюють інстанси сервісів і маперів, тому залежності не жорстко прив’язані. Це дозволяє легко розширювати або замінювати компоненти (наприклад, підключити іншу СУБД або переписати логіку обробки записів) без масштабних змін по всьому проекту. Зрештою, система побудована таким чином, щоб бути гнучкою, модульною та готовою до масштабування.

3.5 Вебклієнт

Реалізація клієнтської частини базується на технологічному стеку React, Zustand, Recharts, Ky, а також загальному принципі розбиття застосунку на модульні сторінки, сервіси та спеціалізовані компоненти.

3.5.1 Архітектура фронтенду

На найвищому рівні архітектура починається з точки входу, яка ініціалізує рендеринг застосунку в браузері (додаток Б.4.1). Тут відбувається обгортання всього застосунку в провайдер маршрутизації, що забезпечує внутрішню навігацію між сторінками без повного перезавантаження. Таким чином реалізується односторінкова архітектура, в якій логіка маршрутизації делегується бібліотеці React Router. Компонент App (додаток Б.4.2) визначає маршрути відповідних сторінок, які відповідають за візуалізацію окремих функціональних блоків системи. До таких сторінок належать журнал логів,

сторінка графіків та інтерфейс налаштувань. Окрім цього, компонент App ініціалізує асинхронне отримання початкових даних (логів та налаштувань) з сервера API, керуючи цим через стан Zustand.

Сторінкова архітектура реалізована за принципом поділу відповідальностей. Кожна сторінка, наприклад LogsPage або ChartsPage, відповідає за певний аспект візуалізації даних. Вони відображають компоненти, які виконують безпосередній рендеринг інтерфейсу, взаємодіють зі сховищем даних для отримання чи оновлення стану, та можуть містити додаткові елементи керування. Цей підхід дозволяє ізолювати логіку окремих частин інтерфейсу, спрощуючи підтримку та масштабування коду.

Бібліотека Zustand забезпечує керування глобальним станом. Архітектура стану поділена на окремі логічні модулі. Наприклад, useLightRecordsStore (додаток Б.4.3) відповідає за логіку, пов’язану з логами світлових записів. Він зберігає сторінку, кількість записів на сторінці, список записів, а також надає функції для пагінації та оновлення даних через API. Подібним чином useSettingsStore керує налаштуваннями системи, дозволяючи читання та оновлення конфігурацій сенсора TCS34725, а також встановлює інтервал потокового логування.

Для зручного управління завантаженням застосовується useAppStore (додаток Б.4.4), який містить булеве значення isLoading та функцію setLoading. Щоб уникнути дублікації коду, для асинхронних операцій створено функцію вищого порядку withLoading, функціональність якої дозволяє централізовано керувати логікою відображення індикатора завантаження без втручання у внутрішню логіку інших сховищ або компонентів.

Значну роль в архітектурі займає шар сервісів, що інкапсулює взаємодію з API. Цей рівень реалізований як об’єкти-класи, кожен з яких наслідує спільний абстрактний клас ApiService (додаток Б.4.5). Цей базовий клас створює екземпляр бібліотеки ky, який попередньо конфігурований з

префіксним URL-адресом до сервера, а також встановлює заголовки для сумісності з тунелюванням через ngrok. На його основі створено окремі сервіси: LightRecordService (відповідає за отримання світлових записів з урахуванням пагінації) та SettingsService (робота з налаштуваннями сенсорів і частоти логування). Розробка сервісів ведеться з дотримуванням принципів інкапсуляції та повторного використання. Наприклад, SettingsService містить методи для оновлення конфігурацій сенсора та параметрів логування, а також для їх отримання. Він оперує структурами, які відповідають інтерфейсам Sensor і Logging, що визначаються в окремих файлах директорії src/domain/models. Це дозволяє однозначно типізувати вхідні та вихідні дані методів, що зменшує ймовірність помилок та спрощує взаємодію з API.

3.5.2 Візуальні компоненти

Компонентна модель інтерфейсу побудована на основі розділення на логічні, функціональні та візуальні блоки. Наприклад, компонент LogsTable відповідає за візуалізацію табличної форми даних логів, реалізуючи сортування, пагінацію, відображення порожнього стану. Графічний компонент Chart забезпечує побудову гістограм на основі Recharts, даючи змогу змінювати параметри осі Y. Обидва компоненти працюють з глобальним станом useLightRecordsStore, отримуючи дані для відображення. Таким чином зміна стану автоматично оновлює візуалізацію.

Окрему категорію становлять компоненти взаємодії з користувачем у формі налаштувань (SensorsForm, LoggingForm). Вони реалізують інтерфейси введення, прив’язані до локального стану (через useState), що ініціалізується поточними значеннями з глобального стану. Після взаємодії користувача ці значення можуть бути відправлені до сервісу через відповідні функції сховища. Підтвердження успішності операції відображається через підключення спеціального хука usePopupTooltip, який активує тимчасове повідомлення-підказку.

Важливими елементами UX-архітектури є компоненти загального користування, такі як ModalWindow (додаток Б.4.6), компонент модального вікна, який приймає заголовок та тіло у вигляді React вузлів, а також прапорець відкриття. Він реалізує логіку закриття вікна через клавішу Escape, а також забезпечує візуальне оформлення з затемненням фону та стилізованим блоком. Компонент Header реалізує статичну шапку сайту з навігаційними вкладками. Він використовує NavLink з бібліотеки маршрутизації, щоб підсвічувати активну вкладку. Завдяки цьому реалізовано інтуїтивну та зручну навігацію між сторінками.

Реалізація візуальних сповіщень побудована за допомогою Tooltip компонента, який за замовчуванням прихований, але активується через спеціальний хук, реалізований як usePopupTooltip. Він дозволяє показувати короткочасне повідомлення в нижній частині екрана без перешкод для основної взаємодії. Такий механізм є ефективним способом зворотного зв’язку при збереженні налаштувань або інших важливих діях.

Таким чином, архітектура фронтенд-частини системи є багаторівневою, модульною та глибоко типізованою. Вона об'єднує сторінкову логіку, централізоване керування станом, сервіси для абстракції запитів API, реактивні компоненти візуалізації даних та зручні інструменти зворотного зв’язку з користувачем.

3.6 Менеджмент проекту

Для організації розробки та забезпечення ефективного контролю за виконанням завдань у проекті використовується система керування задачами.

Канбан-дошка (рисунок 3.5) поділена на чотири основні колонки, які відображають різні етапи життєвого циклу задач: backlog, sprint, in progress, done, який включає в себе перелік завершених задач, що вже інтегровані та документовані.

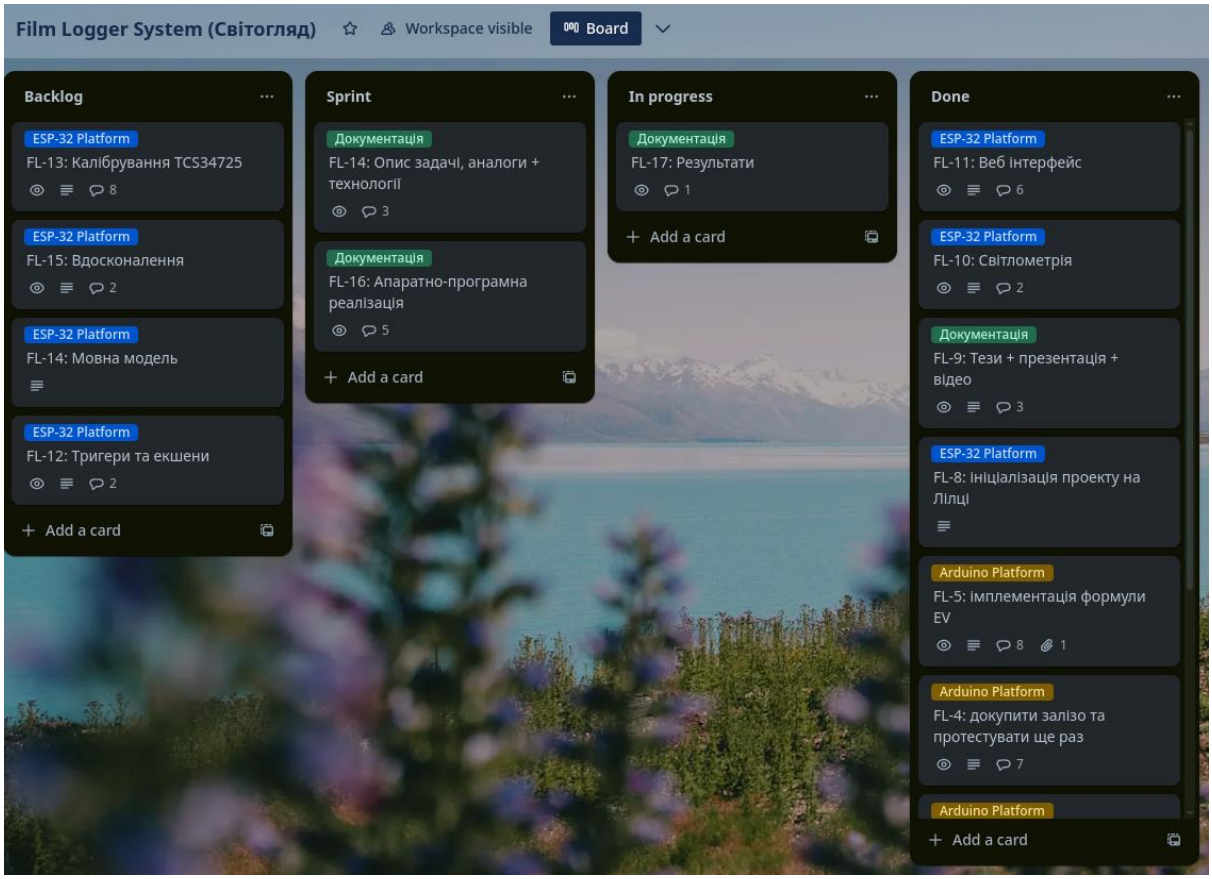

Рисунок 3.5 – Дошка на Trello

Задачі поділені згідно платформ реалізації: ESP32 та Arduino. Також існує відповідний тег для позначки того, що задача відноситься до дослідження або документації. Це дозволяє структуровано розділяти відповідальність між різними завданнями при роботі над системою.

Кожна карточка має унікальний ідентифікатор (наприклад, FL-14), короткий опис, а також активності (коментарі, вкладення тощо). При роботі з системою контролю версій (Git) ідентифікатор задачі ставиться на початок опису кожного коміту, що сприяє прозорому моніторингу прогресу та орієнтуванню в останніх змінах в кодовій базі.

4 РЕЗУЛЬТАТИ РОБОТИ

4.1 Прототип на Arduino

Початкове тестування прототипу системи, реалізованого на платформі Arduino, здійснювалося емпіричним шляхом із застосуванням методів порівняльного аналізу. Зокрема, для валідації вимірювань освітленості, що надходять із цифрового сенсора, використовувався мобільний застосунок LightMeter, встановлений на смартфоні. Порівняння показників освітленості дозволило зробити попередній висновок щодо відповідності вимірювань. Причиною наявних розбіжностей, ймовірно, є відмінність у куті огляду сенсорів: під прямим освітленням (наприклад, при спрямуванні променя від штучного джерела світла безпосередньо на сенсор) результати вимірів для обох пристроїв наближалися на спільного значення.

Слід також зазначити, що при тестуванні не здійснювалося точне калібрування внутрішніх параметрів сенсора, таких як підсилення (gain) та час інтеграції (integration time). Відсутність цього калібрування могла вплинути на коректність отриманих даних, особливо при змінних умовах освітлення.

Також була проведена перевірка ультразвукового сенсора відстані (HC-SR04), результати якого також піддавалися валідації шляхом прямого порівняння з еталонними значеннями, виміряними звичайною лінійкою. Виявлено систематичне зміщення вимірювань приблизно на 3-4 см, яке вимагає врахування під час побудови математичної моделі або подальшого калібрування. Слід зазначити, що величина зміщення є стабільною, що відкриває можливість для його програмної компенсації на рівні прошивки.

Тестування на ранньому етапі розвитку системи дозволило виявити основні особливості та обмеження сенсорної частини, що надає достатньо даних для наступного етапу роботи.

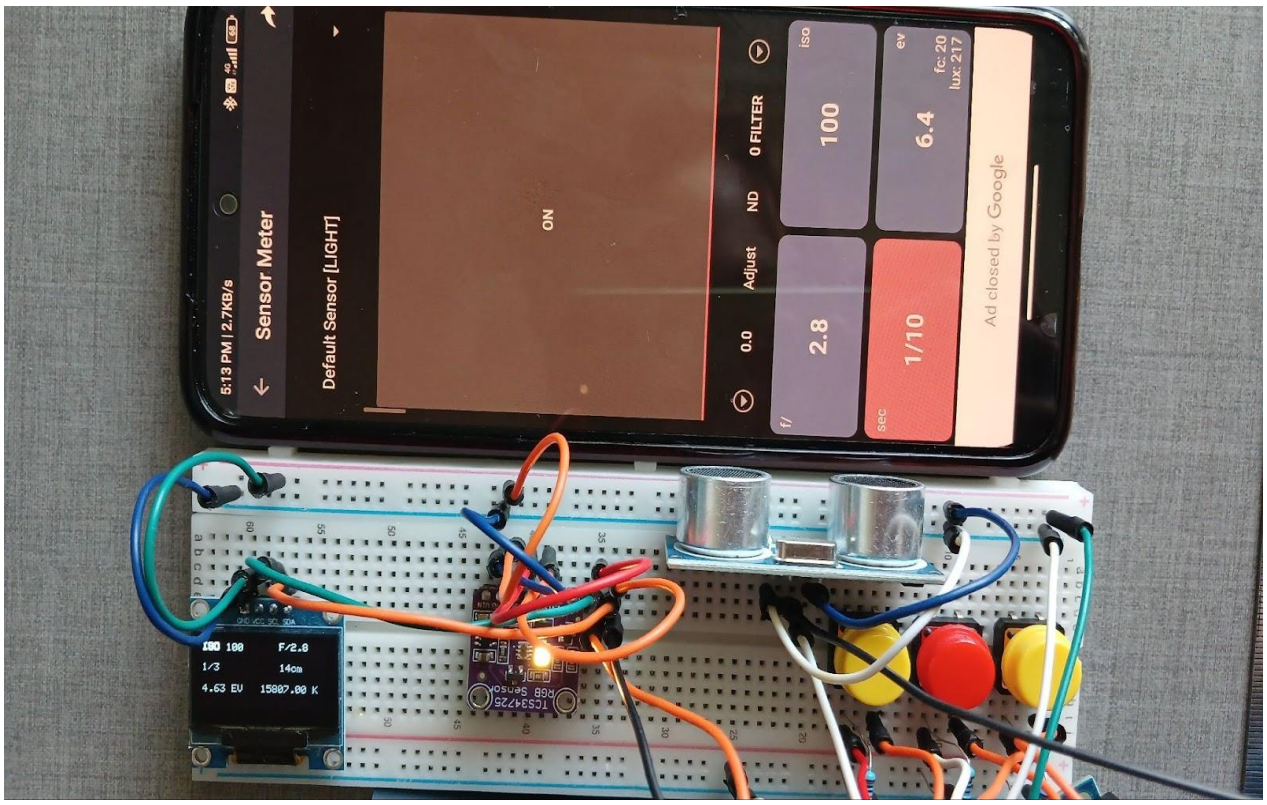

Рисунок 4.1 – Порівняльне тестування функції вимірювання освітлення та розрахунку EV (використання сенсору телефону)

На рисунку 4.1 наведено приклад процесу вимірів освітлення розробленим прототипом, порівнюючи з результатами функції Sensor Meter застосунку LightMeter, яка використовує вбудований сенсор телефону для зміни яскравості екрану.

Тестування проводилося на однакових параметрах ISO та діафрагми, проте найголовнішим показником (незалежним від фотографічних налаштувань) є Exposure Value (EV). Застосунок LightMeter показує значення exposure value яке дорівнює 6.4, прототип на Arduino 4.63.

Далі для порівняння була використана камера телефону (режим Camera Meter застосунку LightMeter, рисунок 4.2). Умовна сцена зйомки була змінена ближче до денного освітлення, проте показники також відрізняються (9.4 EV у застосунку та 5.6 на Arduino). Це також обумовлюється неідеальністю умов тестування, незручністю використання у вигляді зібраного на макетній платі прототипу та різним кутом огляду сенсорів.

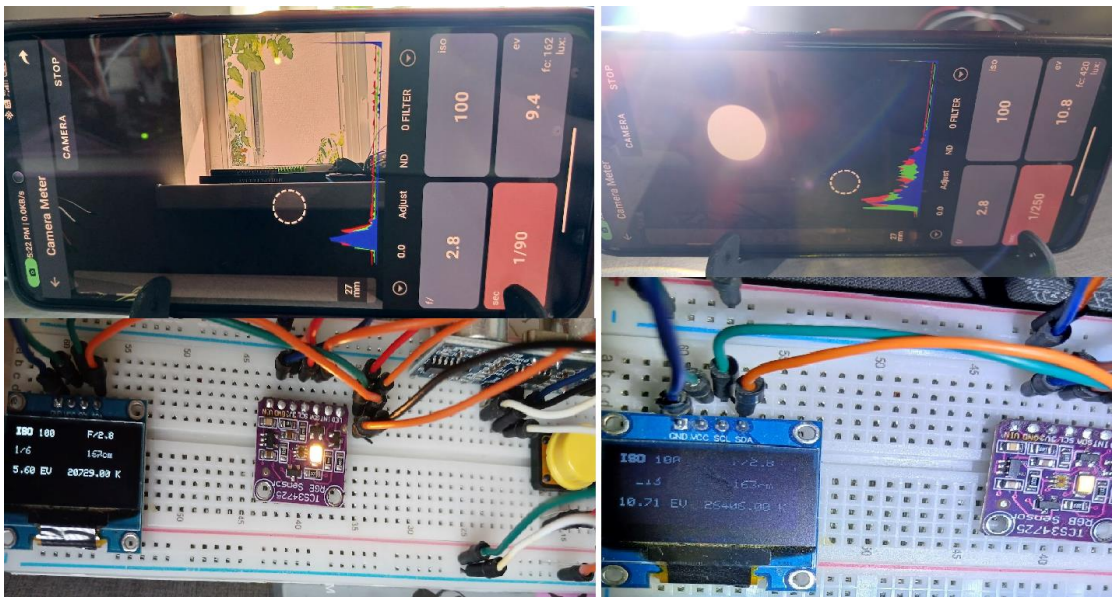

Рисунок 4.2 – Порівняльне тестування вимірювання освітлення та розрахунку EV (при денному освітленні та світлом ручного ліхтарика)

Однак, тестування при прямому стабільному освітленні з майже однакової відстані (рисунок 4.2) до сенсорів дало кращі результати: 10.8 EV в застосунку та 10.71 EV на прототипі Arduino. Це може свідчити про необхідність динамічного калібрування сенсора TCS34725 для пристосування до всіх можливих умов освітлення. Недоліком цього підходу є необхідність постійного перепрошивання мікроконтролера для зміни параметрів gain та integrationTime, що усувається в наступній версії системи.

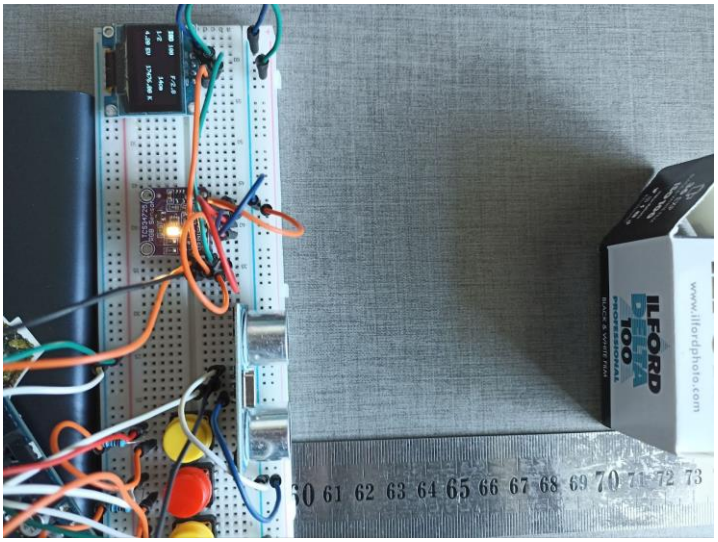

Рисунок 4.3 – Тестування ультразвукового сенсора

На рисунку 4.3 наведено приклад тестування сенсора вимірювання дистанції. Емпіричним шляхом було виявлено похибку в 3-4 сантиметри на дистанціях до 1 метру, що може бути усунено програмним шляхом без втрати якості вимірювань. Це може свідчити про недосконаліть конкретного екземпляру сенсора, накопичувальні похибки при розрахунках відстані або особливість взаємодії ультразвукового сигналу з типами поверхні об’єктів на яких проводилися тести.

4.2 Розгортання API для тестування прототипу основі Лілка

Для інтеграційного тестування системи виникла потреба у розгортанні сервера API в публічно доступному середовищі. Це необхідно для забезпечення повноцінної взаємодії платформи Лілка із сервером, розгорнутому у локальному середовищі розробки (localhost). З огляду на архітектурні особливості та обмеження щодо хостингу, для реалізації такого підходу було використано утиліту ngrok, що дозволяє створювати захищені тунельні з’єднання до локального хоста. Це рішення забезпечило миттєвий доступ до REST API з будь-якого пристрою, підключеного до Інтернету, без необхідності деплойменту на зовнішньому сервері або зміни конфігурації маршрутизатора.

Для ініціалізації сервера було використано команду npm run dev & ngrok http http://localhost:5000 --scheme http. Цей інструкційний рядок виконує послідовно два процеси: запуск локального серверу в режимі розробки (npm run dev) та відкриття HTTP тунелю через ngrok, що дозволяє звертатися до сервера за унікальним URL-адресом. Сумісність з протоколами HTTP/HTTPS виявилась критичною при інтеграції з апаратною платформою Лілка. Через специфіку роботи із TLS-сертифікатами було прийнято рішення не використовувати HTTPS тунель. Таким чином, схема з’єднання була явно вказана як http, що дозволило уникнути конфліктів під час запитів від модуля Лілки до серверу. Попри це, зазначене обмеження не створює загрози безпеці

у контексті локального середовища, де всі з'єднання відбуваються у контрольованій мережі, а всі дані надсилаються мережею без персоналізації.

Використання ngrok (рисунок 4.4) на цьому етапі розробки надало значну гнучкість у тестуванні, особливо при одночасній роботі над клієнтом, сервером і прошивкою. Такий підхід дозволив забезпечити тестування повного циклу передачі даних – від сенсорів на платформі Лілка до їх перегляду у вебклієнті.

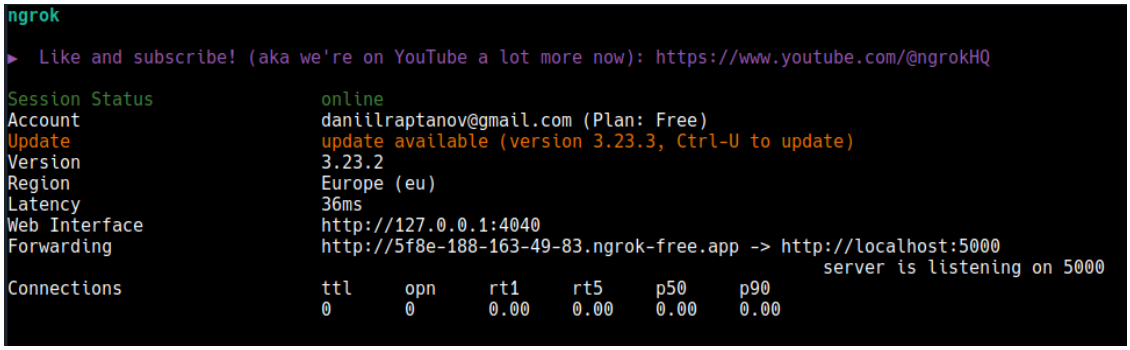

Рисунок 4.4 – Приклад виводу логів тунелювання ngrok

4.3 Прототип на платформі Лілка

Першою дією роботи з платформою Лілка є збірка пристрою згідно офіційній документації [8], що включає в себе монтаж всіх її елементів на печатну плату (рисунок 4.5).

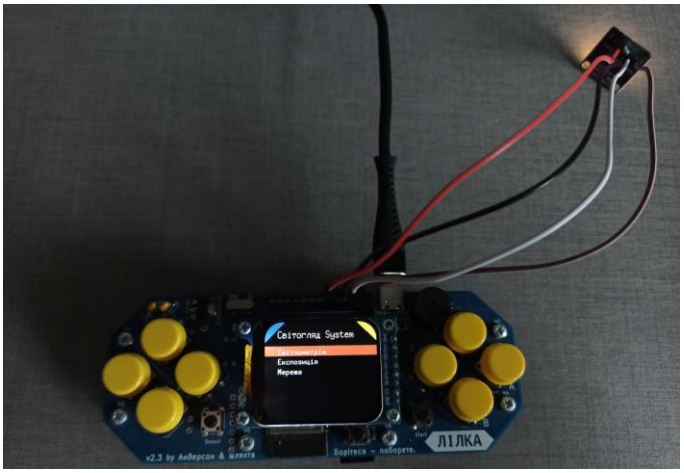

Рисунок 4.5 – Загальний вигляд другого прототипу

Для повноцінної роботи системи логування та аналізу світлового середовища на базі платформи Лілка треба увімкнути тунелювання бекенду через утиліту ngrok. Загальнодоступне посилання разом з параметрами точки доступу WiFi необхідно додати до файлу .secrets.h і наново завантажити прошивку на пристрій. Загальний вигляд пристрою з підключеним до нього сенсором TCS34725 наведено на рисунку 4.5.

Важливим етапом введення в роботу пристрою є підключення сенсору TCS34725. Схема підключення передбачає використання наступних контактів: SDA, SCL, живлення 3.3V та земля GND (рисунок 4.6).

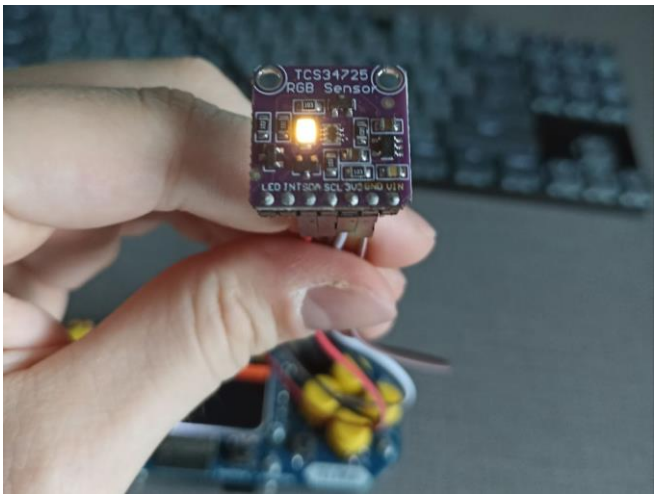

Рисунок 4.6 – Підключений до пристрою сенсор TCS34725

Безпосередньо до пристрою сенсор підключається через роз’єм розширення. Живлення (3.3V) та землю (GND) слід підключати до відповідних пінів на платі. Контакти SDA та SCL підключаються до 14 та до 13 пінів відповідно. Проте, у випадку неможливості задіяти ці піни, треба замінити їх у прошивці на інші.

Наразі система працює у режимі відлагодження, тому в середовищі PlatformIO (або будь-якій іншій IDE, адаптованої до особливостей embedded розробки) треба обрати опцію Monitor (або Upload and Monitor, якщо прошивка ще не була оновлена). В консолі з’являться логи щодо стану системи одразу після ввімкнення пристрою.

Одразу після ввімкнення, користувачу відображається головне меню системи з пунктами світлометрії, експозиції та мережі (рисунок 4.7). Навігація між пунктами меню здійснюється кнопками Up і Down, а вибір – кнопкою A. Вихід з будь-якого розділу меню та повернення до головного екрану здійснюється кнопкою Start, про що повідомляє підпис знизу екрана в конкретних розділах.

Рисунок 4.7 – Головне меню системи

У розділі світлометрії (рисунок 4.8) користувачу відображаються дві шкали для умовних позначень люксів та колірної температури світла. Шкала люксів складається з градієнту (плавний перехід від чорного до білого) та підпису lux. Шкала колірної температури складається з загальноприйнятої шкали відображення correlated color temperature у Кельвінах (від красного до синього кольору), та позначкою K, що вказує на одиниці виміру (Кельвіни).

Рисунок 4.8 – Інтерфейс світлометрії

Інтерфейс побудований таким чином, що при граничних значеннях люксів або колірної температури написи про конкретні значення все одно будуть видимими на екрані (рисунок 4.9).

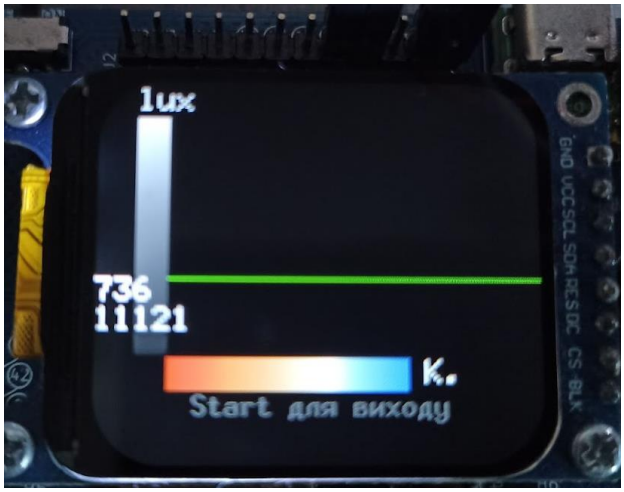

Рисунок 4.9 – Відображення нетипових даних світлометрії

Всі вимірювання сенсором у режимі відлагодження виводяться у послідовний порт в середовищі розробки. Для швидкого розуміння стану системи в тестовому режимі достатньо наступної інформації з сенсора: загальний (чистий) канал Clear, червоний канал Red, зелений канал Green, синій канал Blue та розраховані люкси Lux.

В розділ світлометрії також інтегрована система запису логів на картку пам’яті. Логування відбувається в двох режимах: single та stream. Single режим – це одноразове додавання поточних відомостей щодо експозиції та світлового середовища до файлу, вмикається кнопкою D. Режим stream – це безперервне логування з певним інтервалом (який налаштовується через вебінтерфейс системи), вмикається та вимикається кнопкою C.

Про успіх увімкнення чи вимкнення модуля логування повідомляє п’єзо-динамік, вбудований в корпус Лілки, а також додаткові записи в послідовному порті.

Другий розділ, який доступний через інтерфейс користувача, стосується суто фотографічного поняття експозиції та організований

наступним чином: користувачу доступні для взаємодії лише параметри ISO та діафрагма, яка позначається написом F. За замовчуванням ці значення є нульовими (рисунок 4.10).

Рисунок 4.10 – Розділ налаштувань експозиції

Для навігації між параметрами ISO та діафрагма використовуються кнопки Up і Down. Обраний користувачем параметр підсвічується жовтим кольором. Для зміни значення параметру використовуються кнопки Right і Left (рисунок 4.11).

Рисунок 4.11 – Приклад зміни параметра ISO

Після того як користувач вибирає ненульові значення для обох параметрів, розраховується витримка (третій рядок зверху, позначка 1/). Якщо розрахована витримка не співпадає з рекомендованою (рисунок 4.12) для поточного освітлення – напис відображається червоним або жовтим кольором. В іншому випадку – зеленим.

Рисунок 4.12 – Розрахована витримка не співпадає з рекомендованою

Для візуальної зручності була додана шкала з позначками EV (exposure value) від 0 до 20, зеленими рисками (діапазон рекомендованих значень EV під поточні налаштування параметрів ISO та діафрагми) та червоною рискою – поточним значенням EV. Коли поточне значення EV входить до діапазону рекомендованих значень, можна вважати розраховану витримку коректною для поточного освітлення (рисунок 4.13).

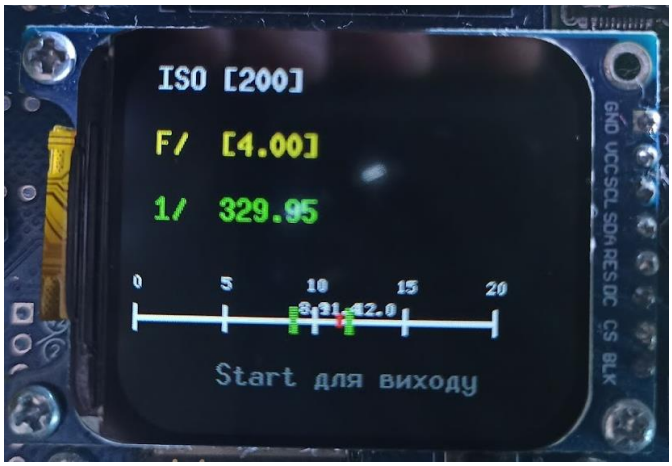

Рисунок 4.13 – Поточне EV включається в діапазон рекомендованого

Рекомендоване EV визначається діапазоном значень, які базуються на використанні фотографічних приладів без застосування штативу (тобто зйомка з рук). Тому, рекомендовані значення витримки для ISO 100 та діафрагми 2.8 буде діапазон від 1/60 секунди до 1/500.

Третій розділ інтерфейсу користувача відповідає за всі дії пов’язані з мережею. При першій спробі після перезавантаження спочатку відбувається підключення до WiFi мережі (рисунок 4.14), ім’я та пароль якої були додані до .secrets.h попередньо.

У випадку помилки підключення або відсутності доступних мереж, користувачу відображається відповідне повідомлення (рисунок 4.14).

Якщо пристрій успішно підключився до мережі WiFi, відбувається спроба під’єднатися до сервера. Спочатку надсилається запит типу ping для перевірки чи працює бекенд взагалі. Якщо відповідь негативна – відображається відповідне повідомлення на екрані (рисунок 4.14).

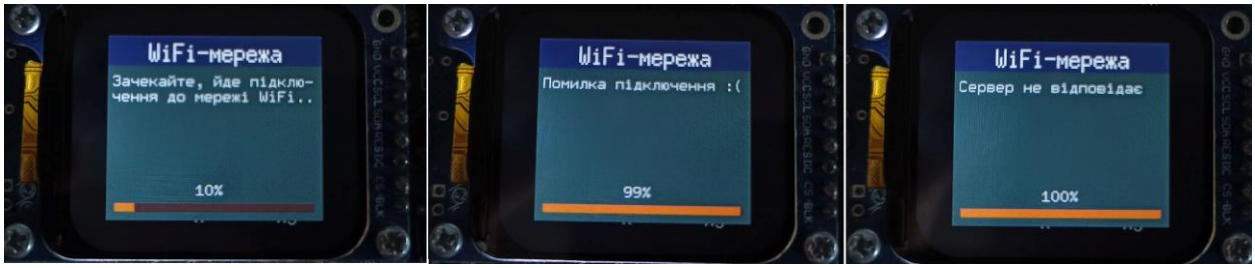

Рисунок 4.14 – Процес під’єднання пристрою до мережі

Якщо відповідь коректна – надсилаються запити на отримання налаштувань пристрою: параметри конфігурації сенсора TCS34725 та інтервал логування в режимі stream. Повідомлення про успіх або невдачу також відображається користувачу (рисунок 4.15).

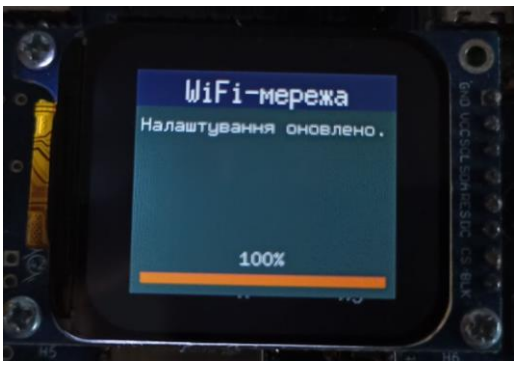

Рисунок 4.15 – Повідомлення про успіх операції оновлення налаштувань пристрою

Далі відбувається перевірка наявності незбережених на віддаленому сервері даних. Якщо такі дані присутні локально на пристрої, відображається їх кількість і одночасно з цим відбувається синхронізація по мережі (рисунок 4.16). Ліміт рядків на один запит є незмінним та встановлено за замовчуванням 10.

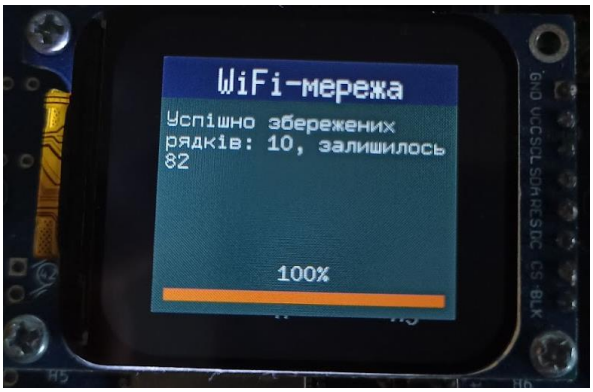

Рисунок 4.16 – Екран синхронізації даних по мережі

Якщо даних для синхронізації не знайдено, або всі дані були синхронізовані, це також повідомляється користувачу (рисунок 4.17).

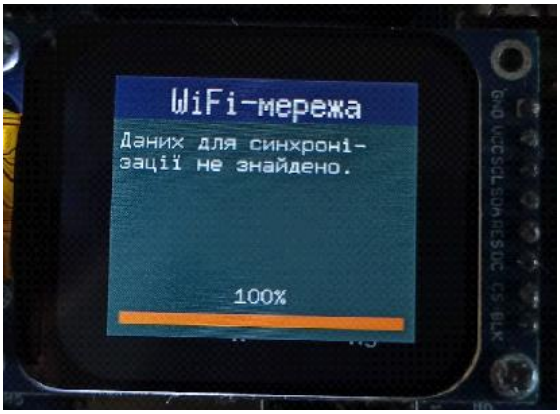

Рисунок 4.17 – Повідомлення про відсутність даних для синхронізації

На етапі розробки дуже важливо вчасно помічати помилки при роботі з мережею як на рівні бекенду, так і на рівні ESP32. Тому всі дії, пов’язані з модулем WiFiConfig логуються в послідовному порті. Всі дані в незалежності від того, чи були вони синхронізовані з бекендом, залишаються збереженими

у файлі fl_logs.csv на карті пам’яті (рисунок 4.18). Дані можна обробляти будь-якими зручними засобами за межами розробленої системи. Рядки легко ідентифікувати за першим стовпцем, де значення представлені у форматі uuidv4, а останній стовпець synced потрібен для розрізнення синхронізованих і не синхронізованих рядків.

	A	B	C	D	E	F	G	H
1	id	iso	aperture	shutter	lux	cct	ev	synced
2	9dcf0896-5834-44ff-9b58-3b5867530597	0	0.00	nan	6.00	1632.00	1.26	1
3	8649ea7e-5f75-482c-874c-f9c8c9790fa3	0	0.00	nan	6.00	1632.00	1.26	1
4	d86d01b8-bfc8-4a27-9833-5cff11f15502	0	0.00	nan	6.00	1632.00	1.26	1
5	c25a31df-add5-4aff-929c-586a5bb89762	0	0.00	nan	6.00	1632.00	1.26	1
6	71b30394-4733-43fd-b6e4-f3bac5c947cc	0	0.00	nan	6.00	1632.00	1.26	1
7	0ddc1bd4-270d-4d98-b5a9-34aba966d90	0	0.00	nan	9.00	1658.00	1.85	1
8	e407a829-6ced-47f0-87ea-3052c3c6c3c7	0	0.00	nan	8.00	1658.00	1.85	1
9	54c4d7d9-1961-47a9-9814-49c850bd619	0	0.00	nan	9.00	1658.00	1.85	1
1198	ac1709be-6061-4799-aba9-cdb5b8a3d378	0	0.00	nan	289.00	32106.00	6.85	1
1199	0a6f5b22-86fe-46eb-9f1f-28401edb1c0e	0	0.00	nan	305.00	61248.00	6.93	1
1200	e294e7b9-b453-483f-a31b-91b5b2900031	0	0.00	nan	305.00	51664.00	6.93	1

Рисунок 4.18 – Збережені дані логування у форматі csv

4.4 Робота вебклієнта

Для роботи вебклієнта необхідно перевірити чи працює бекенд та ngrok тунелювання (рисунки 4.19 та 4.20).

```
ngrok

Make HTTP calls to internal services from your gateway (dev preview): https://ngrok.com/r/http-request

Session Status                online
Account                       danilraptanov@gmail.com (Plan: Free)
Version                       3.23.2
Region                        Europe (eu)
Latency                       36ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://0a984d201231.ngrok-free.app -> http://localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              25      0       0.01    0.03    5.03    5.22

HTTP Requests
-------------

11:07:14.048 EEST GET  /api/v1/logging             200 OK
11:07:13.637 EEST GET  /api/v1/sensors             200 OK
11:07:13.238 EEST GET  /api/v1/system              200 OK
11:04:52.329 EEST GET  /api/v1/sensors             200 OK
11:04:52.746 EEST GET  /api/v1/logging             200 OK
11:04:51.926 EEST GET  /api/v1/system              200 OK
11:04:48.028 EEST GET  /api/v1/system              502 Bad Gateway
11:04:44.139 EEST GET  /api/v1/system              502 Bad Gateway
11:04:40.258 EEST GET  /api/v1/system              502 Bad Gateway
11:03:43.727 EEST POST /api/v1/light-records/import 201 Created
```

Рисунок 4.19 – Логи утиліти ngrok

В режимі розробки було запущено серверну та клієнтську частину WEB на одному комп’ютері, тому змінна API_URL у файлі .env при налаштуванні клієнта була зі значенням localhost:5000.

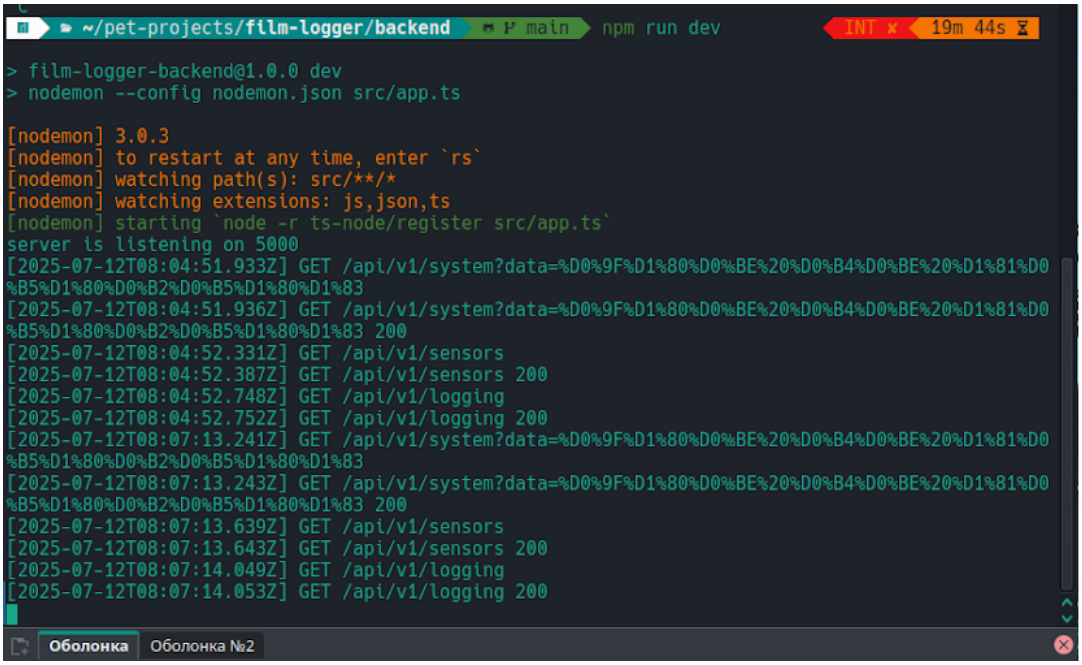

Рисунок 4.20 – Логи серверної частини системи

При необхідності доступу клієнта в Інтернеті слід врахувати особливості роботи ESP32 з захищеним протоколом https. Тому в режимі розробки демонструється спрощена модель розгортання системи, що також додає більше контролю над ситуацією та перегляд логів в режимі реального часу. Для локального розгортання вебзастосунку треба виконати команду npm run dev в терміналі (рисунок 4.21).

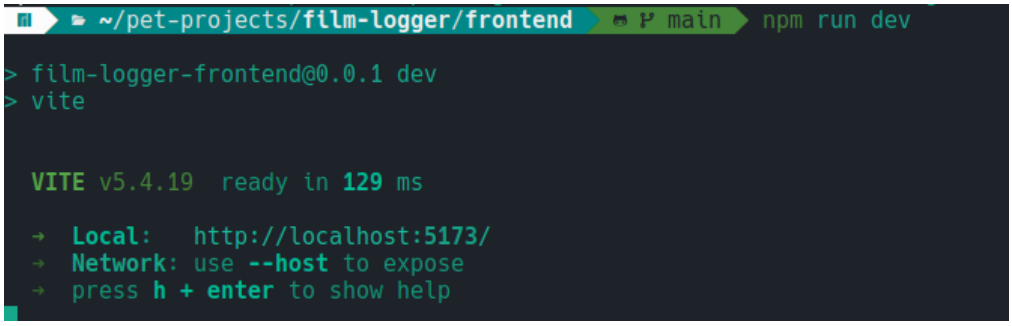

Рисунок 4.21 – Запуск клієнтського вебзастосунку на локальному комп’ютері

Фронтенд частина системи є single-page застосунком з використанням роутингу для зручності перемикання між контентом. Перша вкладка на сайті є відображенням збережених у системі логів. Дані представлені у табличному вигляді з пагінацією та зміною кількості елементів на сторінці (рисунок 4.22).

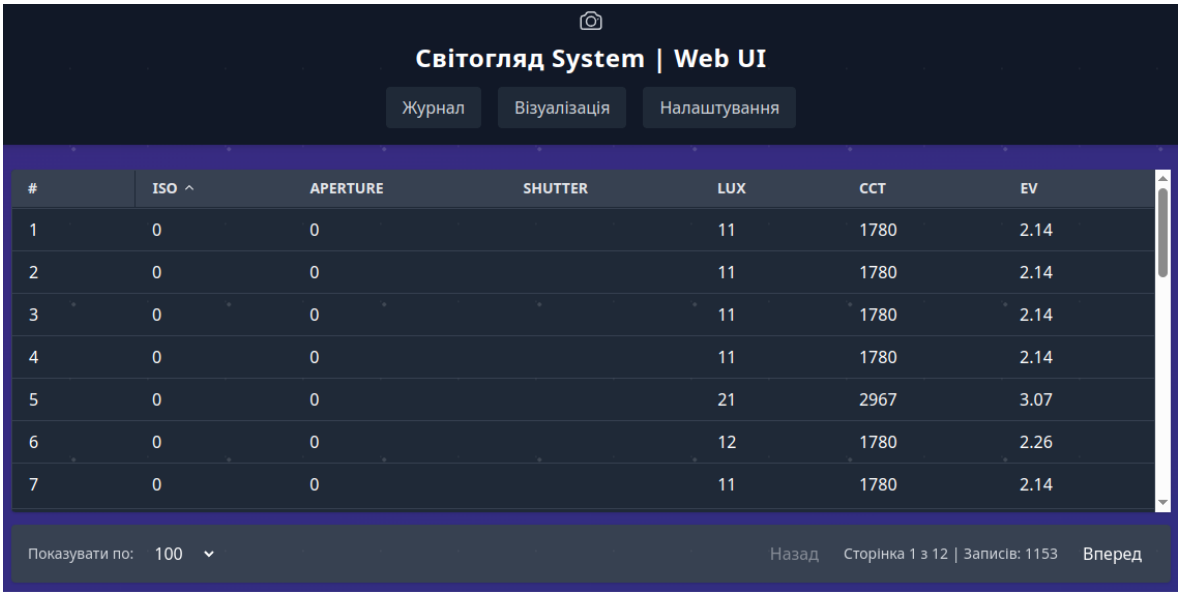

Рисунок 4.22 – Таблиця логів

Для будь-якого односторінкового застосунку важливим є швидке реагування на дії користувача, наприклад, при надсиланні запиту на отримання наступної сторінки в таблиці. Якщо запит виконується довго, або сервер не відповідає, користувачу показується екран завантаження (рисунок 4.23). Емулювання тривалого завантаження даних доступно у консолі розробника в браузері (вкладка Network).

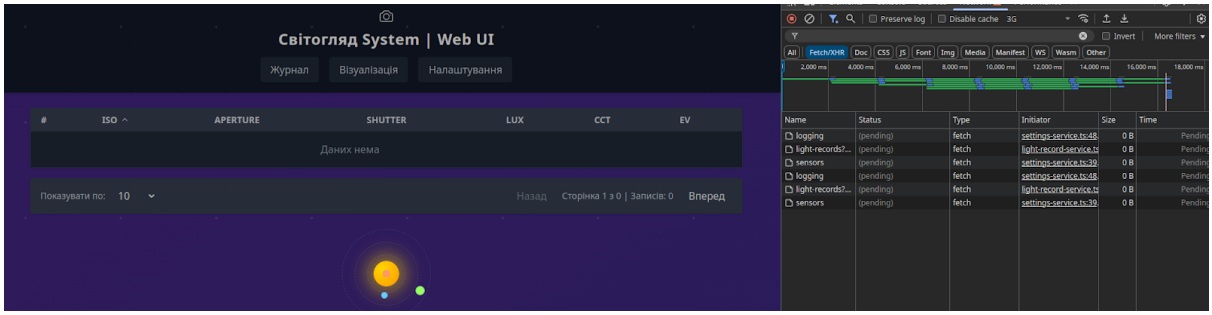

Рисунок 4.23 – Екран завантаження

Важливою складовою вебзастосунків є можливість переглядати контент на малих розширеннях екрану (наприклад, з мобільного телефону), що було враховано при розробки клієнтської частини (рисунок 4.24).

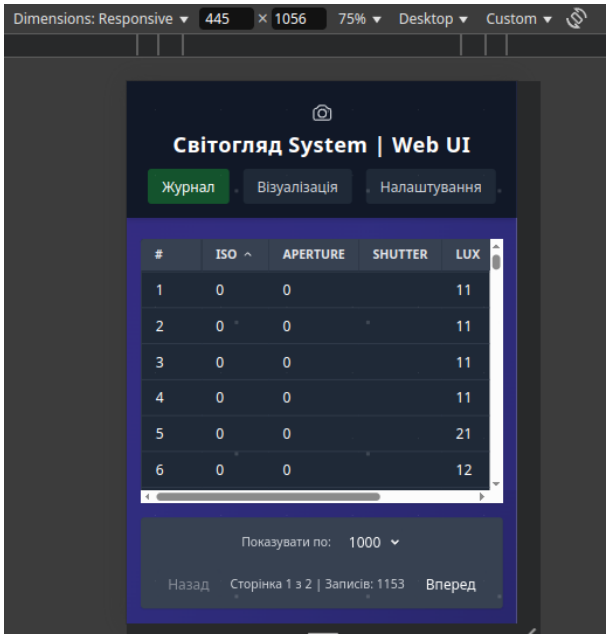

Рисунок 4.24 – Адаптація таблиці під мобільні телефони

Друга вкладка відображає дані, завантажені в таблицю у вигляді графіків (рисунок 4.25). Для підвантаження нових даних треба перейти на вкладку Журнал, вибрати бажану сторінку і повернутися на Візуалізацію для перегляду табличних даних в графічному вигляді.

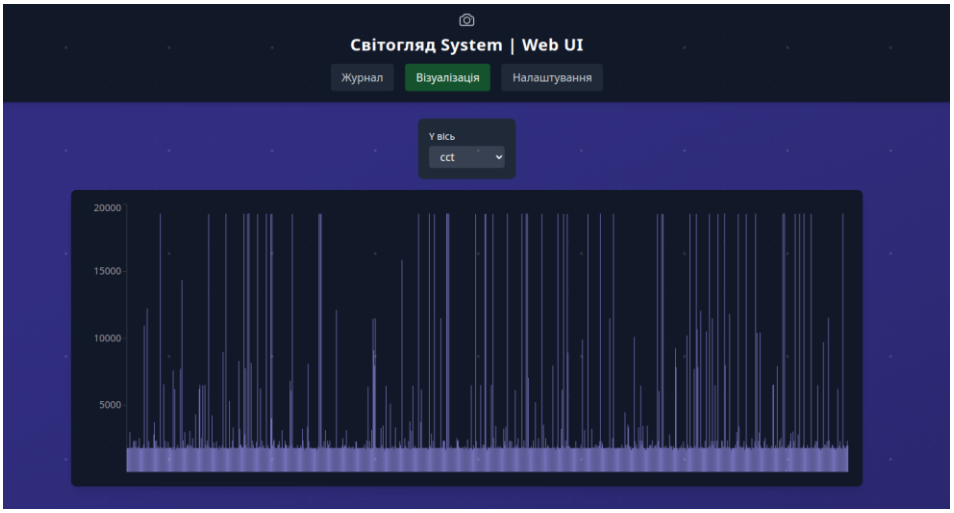

Рисунок 4.25 – Вкладка візуалізації табличних даних у вигляді графіків

Функціонал передбачає зміну атрибуту за яким відбувається візуалізація (вісь Y) за допомогою випадаючого меню (рисунок 4.26).

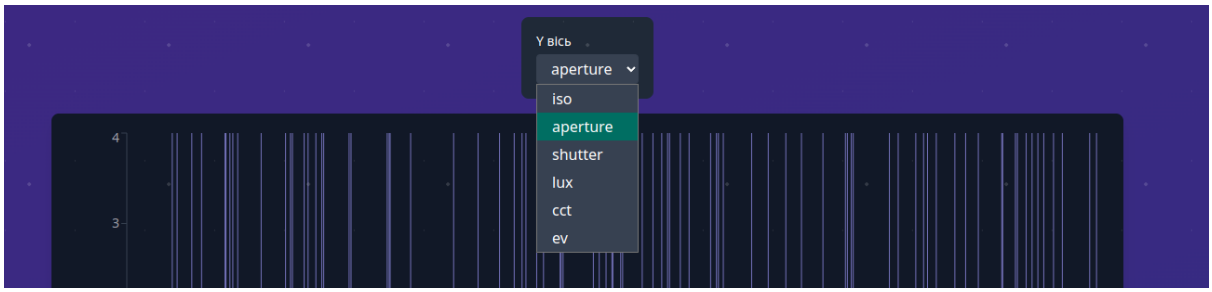

Рисунок 4.26 – Зміна атрибуту за яким будується графік

Третя вкладка інтерфейсу користувача містить дві форми для налаштувань пристрою (рисунок 4.27). Перша форма використовується для зміни параметрів gain та integrationTime сенсора TCS34725, друга – для зміни частоти потокового логування. Поля обмежені для вводу будь-яких символів окрім чисел.

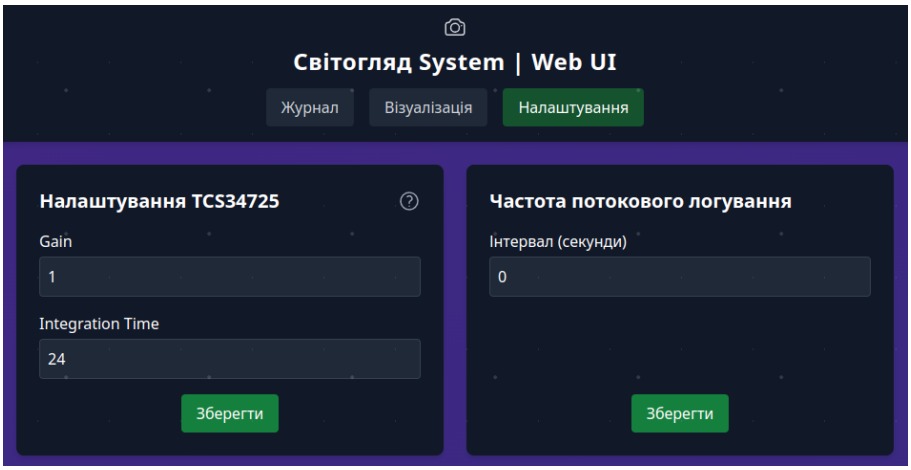

Рисунок 4.27 – Вкладка налаштувань пристрою

Для нагадування користувачу коректних для сенсора TCS34725 даних gain та integrationTime розміщена спеціальна підказка (знак питання) в правому верхньому куті форми. Ця кнопка відкриває модальне вікно з детальною інформацією щодо всіх можливих значень параметрів.

ВИСНОВКИ

У результаті аналітичного огляду предметної області було виявлено, що традиційні фотометричні прилади (люксметри, точкові експонометри тощо) вирішують окремі аспекти завдань світлометрії, але не здатні забезпечити комплексне логування та гнучке зональне і спектральне вимірювання. Аналіз існуючих рішень показав можливість застосування інтегральних експонометрів, проте їхній функціонал обмежений і вимагає розширення у напрямку цифрового збереження даних та забезпечення відкритості інтеграції.

У якості технологічного стеку обрані апаратна платформа Lilka на базі ESP32, що забезпечує надійний збір та попередню обробку даних; бекенд на Node.js у поєднанні з PostgreSQL (надають можливість масштабування й структуроване зберігання); односторінковий React-клієнт – інтуїтивний доступ до візуалізації, налаштувань і аналітики. Концепція модульності та відкритості дозволяє поступово інтегрувати нові сенсори, аналітичні алгоритми та механізми обробки подій без порушення сумісності.

Таким чином, поєднання результатів дослідження предметної області та вибраних технологічних підходів забезпечило основу для розробки універсального інструменту для вимірювання, логування та адаптивного аналізу світлового середовища яке може бути застосованим у багатьох сферах, зокрема у мистецтві фотографії.

Результати роботи були представлені на виставці технічної творчості на 29-му Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» (представлення експонату відзначено нагородою за 2 місце) а також в рамках п’ятнадцятої міжнародної науково-технічної конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» [34].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Li C., Guo C., Han L., Jiang J., Cheng M.-M., Gu J. Low-Light Image and Video Enhancement Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2022. Vol. 44, No 12. P. 9396-9416. DOI: 10.1109/TPAMI.2021.3126387

2. Stefani O., Schöllhorn I., Münch M. Towards an evidence-based integrative lighting score: a proposed multi-level approach. *Annals of Medicine*. 2023. No 56. P. 1-16. DOI: https://doi.org/10.1080/07853890.2024.2381220

3. Kong Z., Liu Q., Li X., Hou K., Xing Q. Indoor lighting effects on subjective impressions and mood states: A critical review. *Building and Environment*. 2022. Vol. 224. P. 1-46. DOI: 10.1016/j.buildenv.2022.109591

4. Tsesmelis T., Hasan I., Cristani M., Galasso F., Del Bue A. RGBD2lux: Dense Light Intensity Estimation With an RGBD Sensor. *IEEE Winter Conference on Applications of Computer Vision*. 2019. P. 501-510.

5. Spitschan M., Hammad G., Blume C., Schmidt C., Skene D.J., Wulff K., Santhi N., Zauner J., Münch M. Metadata recommendations for light logging and dosimetry datasets. *BMC Digital Health.* 2024. Vol. 2, No 73. P. 1-10.

6. Cords P. H., Fatora D. A. Relating the Ansel Adams Zone System to the ASA Criteria for the Determination of Film Speeds. *Optical Engineering.* 1964. Vol. 2, No 6. DOI: https://doi.org/10.1117/12.7971290

7. Wheeler P. Practical Cinematography. 2nd Edition, New York, 2005. 224 p. DOI: https://doi.org/10.4324/9780080480152

8. Документація проєкту Lilka. URL: https://docs.lilka.dev/uk/latest. (дата звернення: 12.06.2025).

9. Tabaka P., Wtorkiewicz J. Analysis of the Spectral Sensitivity of Luxmeters and Light Sensors of Smartphones in Terms of Their Influence on the Results of Illuminance Measurements – Example Cases. *Energies*. 2022. No 15. P. 1-21. DOI: https://doi.org/10.3390/en15165847

10. BH1750 Light Sensor Module. URL: https://www.handsontec.com/dataspecs/sensor/BH1750%20Light%20Sensor.pdf. (дата звернення:12.05.2025).

11. TSL2591 Datasheet. URL: https://cdn-shop.adafruit.com/datasheets/TSL25911_Datasheet_EN_v1.pdf. (дата звернення: 12.06.2025).

12. Pan D., Jiang Z., Li Y., Yu H., Gui W. A Novel Compensation Method for Infrared Temperature Measurement Using Infrared Vision and Visible Light Vision Under Water Mist Interference. *IEEE Transactions on Instrumentation and Measurement.* 2023. Vol. 72, No 4506809. P. 1-9.

13. AS7262 6-Channel Visible Spectral Device with Electronic Shutter and Smart Interface. Description. URL: https://www.mouser.com/datasheet/2/588/AS7262_DS000486_2-00-1082195.pdf. (дата звернення: 12.06.2025).

14. TCS34725 Color light-to-digital converter. URL: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf. (дата звернення: 12.06.2025).

15. Tominaga S., Ebisui S., Wandell B. A. Color Temperature Estimation of Scene Illumination. *Color and Imaging Conference*. 1999. Vol. 7, No 9. P. 42-47. DOI: https://doi.org/10.2352/CIC.1999.7.1.art00009

16. Strong S. Photometry. *Introduction to Visual Optics*, 2024. No 12. P. 113-122. DOI: https://doi.org/10.1016/B978-0-323-87534-9.00021-0

17. Setting Camera Exposure URL: http://www.dougkerr.net/Pumpkin/articles/Setting_Exposure_Ev.pdf. (дата звернення: 24.05.2025).

18. Photometry – The CIE system of physical photometry. URL: https://www.iso.org/standard/83178.html. (дата звернення: 24.05.2025).

19. Lee H. C. Introduction to Color Imaging Science. Foxlink Peripherals Inc. New York : Cambridge University Press, 2005. 695 p.

20. Bjelkhagen H. I. Silver halide Recording Materials. Springer Series in Optical Sciences. Heidelberg, 1995. 440 p.

21. Jacobson R. E. The Manual of Photography: Photographic and Digital Imaging. 9th Edition. Oxford : Focal Press, 2000. 496 p.

22. Rees W. G. Physical Principles of Remote Sensing. Second Edition. New York : Cambridge University Press, 2001. 360 p.

23. Peterson B. Understanding Exposure: How to Shoot Great Photographs with a Film or Digital Camera. New York : Amphoto Books, 2004. 160 p.

24. Jacobson R., Ray S., Attridge G., Axford N. The Manual of Photography. Boston : Focal Press, 2000. 464 p.

25. Langford M., Fox A, Smith R. Basic Photography. Seven Edition. Boston : Focal Press, 2000. 368 p.

26. Sheppard R. Digital Photography: Top 100 Simplified Tips & Tricks. Fourth Edition. Indianapolis : Visual/Wiley, 2010. 227 p.

27. Lynch J., Barbara A., Perkins M. Illustrated Dictionary of Photography. NewYork : Amherst Media, 2008. 141 p.

28. Hullfish S., Fowler J. Color Correction for Digital Video. Boston : Focal Press, 2002. 202 p.

29. Jackman J. Lighting for Digital Video & Television. Second Edition. Boston : Focal Press, 2004. 256 p.

30. George, C. Total Digital Photography. Philadelphia : Running Press, 2006. 320 p.

31. Child J., Galer M. Photographic Lighting: Essential Skills. Fourth Edition. Boston : Focal Press, 2008. 216 p.

32. Busch D. Mastering Digital Photography: The Photographer's Guide to Professional Quality Digital Photography. Second Edition. Boston : Thomson Course Technology, 2003. 310 p.

33. Photometry: The Answer to How Light Is Perceived. URL: https://www.photonics.com/Articles/Photometry_The_Answer_to_How_Light_Is_Perceived/a25119/ (дата звернення: 20.05.2025).

34. Раптанов Д., Іващенко Г. Засоби вирішення проблеми аналізу умов зйомки для плівкової фотографії. *Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління.* 2025. Т. 1. С. 138. DOI: https://doi.org/10.32620/ICT.25.t1