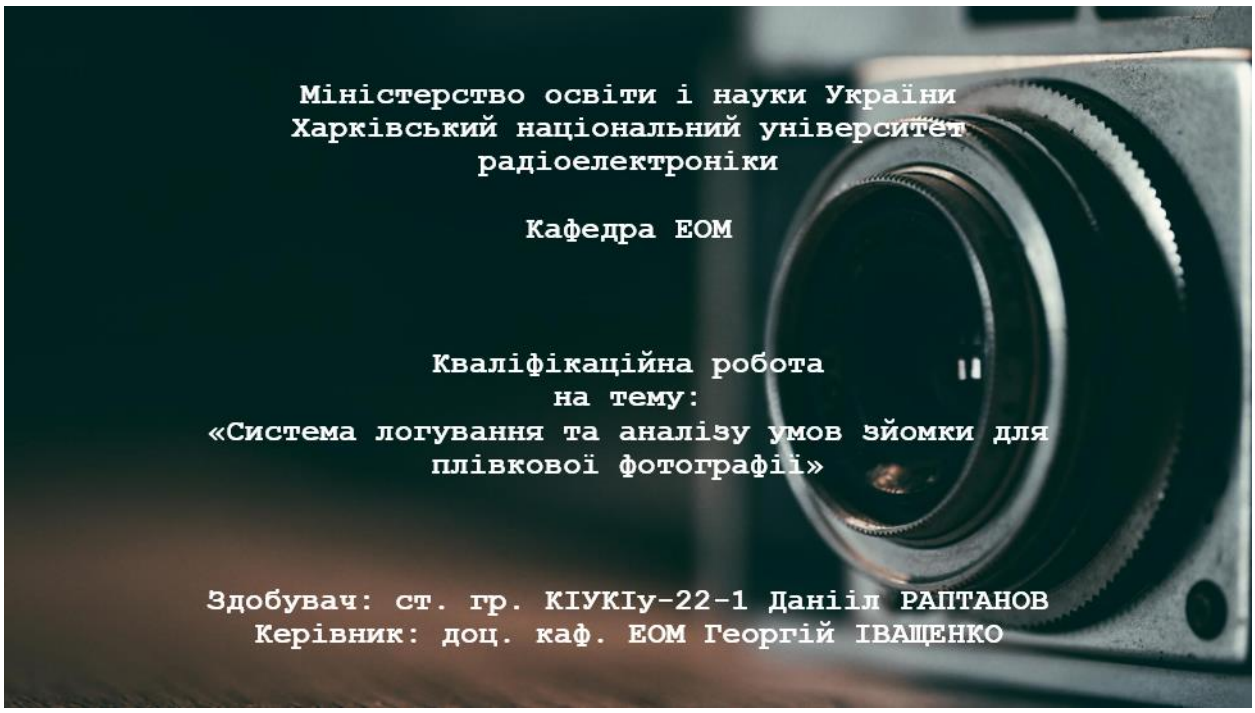
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи



Актуальність

- Процес фотографії вимагає вимірювання параметрів світлового середовища
- Вимірювання параметрів світла наявними засобами має недостатню точність
- Відсутність можливості використання зібраних даних для аналізу
- Необхідність точного відтворення умов експозиції

Існуючі рішення

Люксометр – **простий** прилад для **швидкого** вимірювання **загальної** освітленості.

Точковий експонометр – більш розповсюджений серед професійних фотографів прилад, що має **функцію зонального вимірювання.**

Інтегральний експонометр – забезпечує усереднення виміру експозиції по всій площі кадру, швидке визначення **базової** експозиції, **вбудований** у більшість камер.

Застосунок LightMeter – **мобільний** застосунок, використовує засоби смартфону (камеру чи сенсори) для отримання інформації щодо навколишнього середовища.

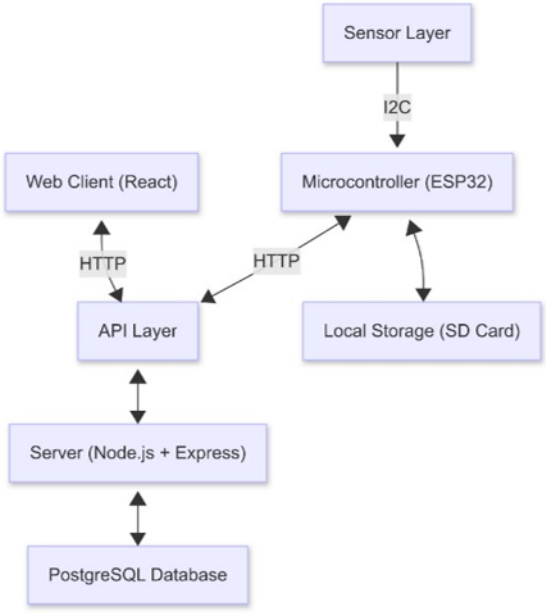
3

Постановка задачі

Перевірити можливість централізованого збору даних про світло, використовуючи наявні апаратні засоби.

Реалізувати модульну систему логування та аналізу параметрів світлового середовища.

Розгорнути клієнт-серверний застосунок для накопичення, синхронізації, перегляду та аналізу зібраних даних щодо умов зйомки.



4

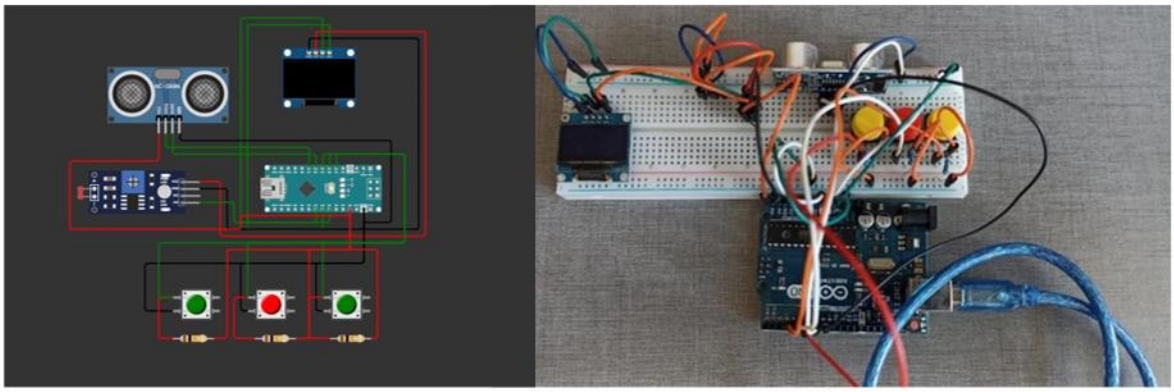
Основні апаратні засоби

5

Використані програмні засоби

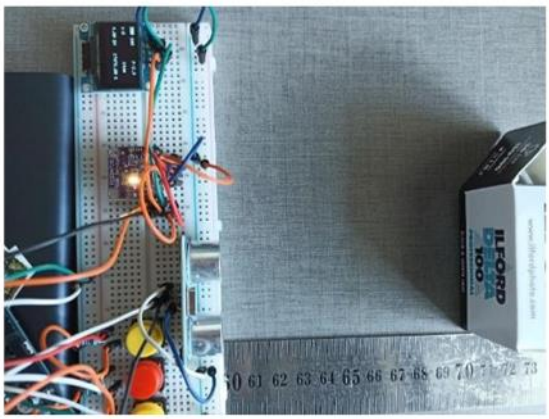
6

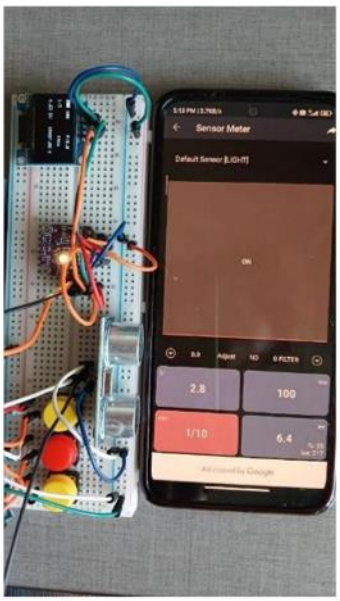
Прототип на Arduino



7

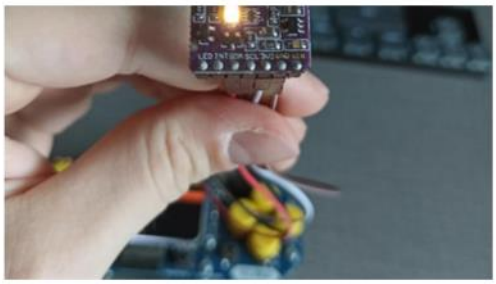
Тестування прототипу на Arduino

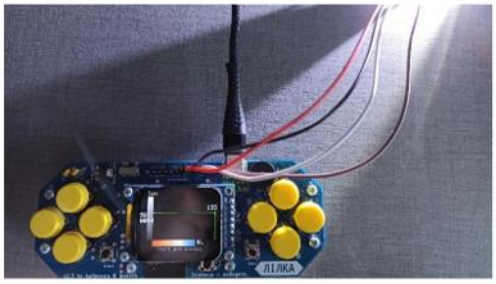


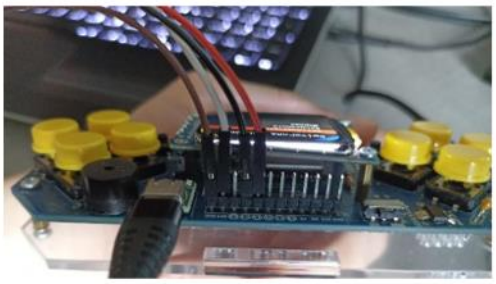


8

Підключення та використання сенсорів

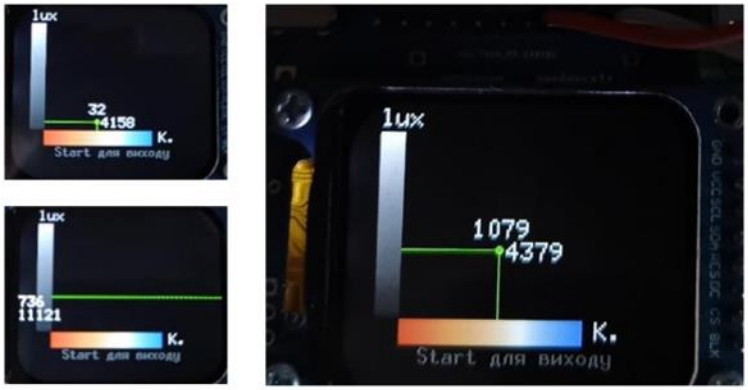






10

Приклади результатів світлометрії

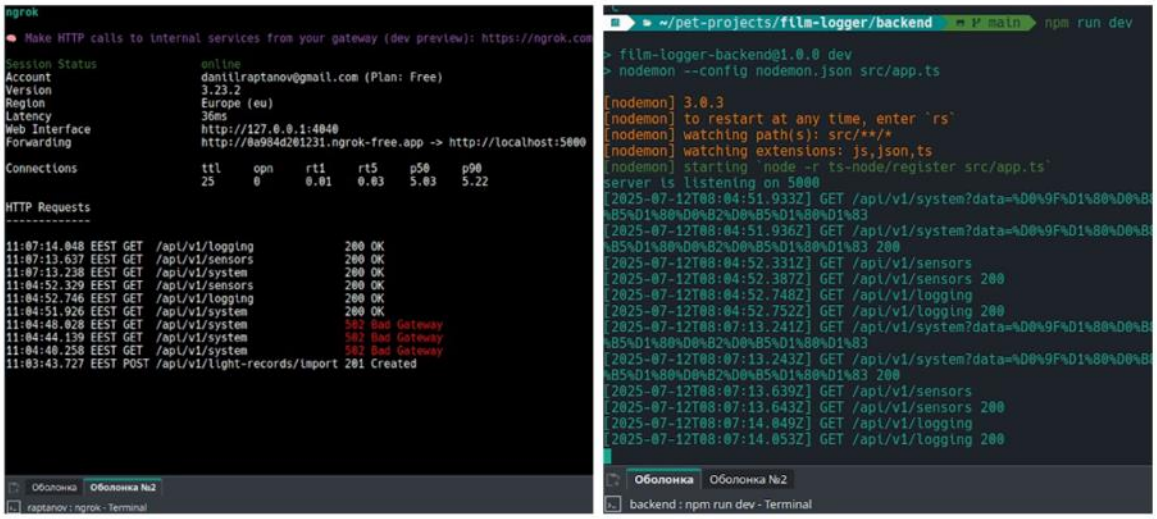


11

Виведення результатів розрахунку експозиції

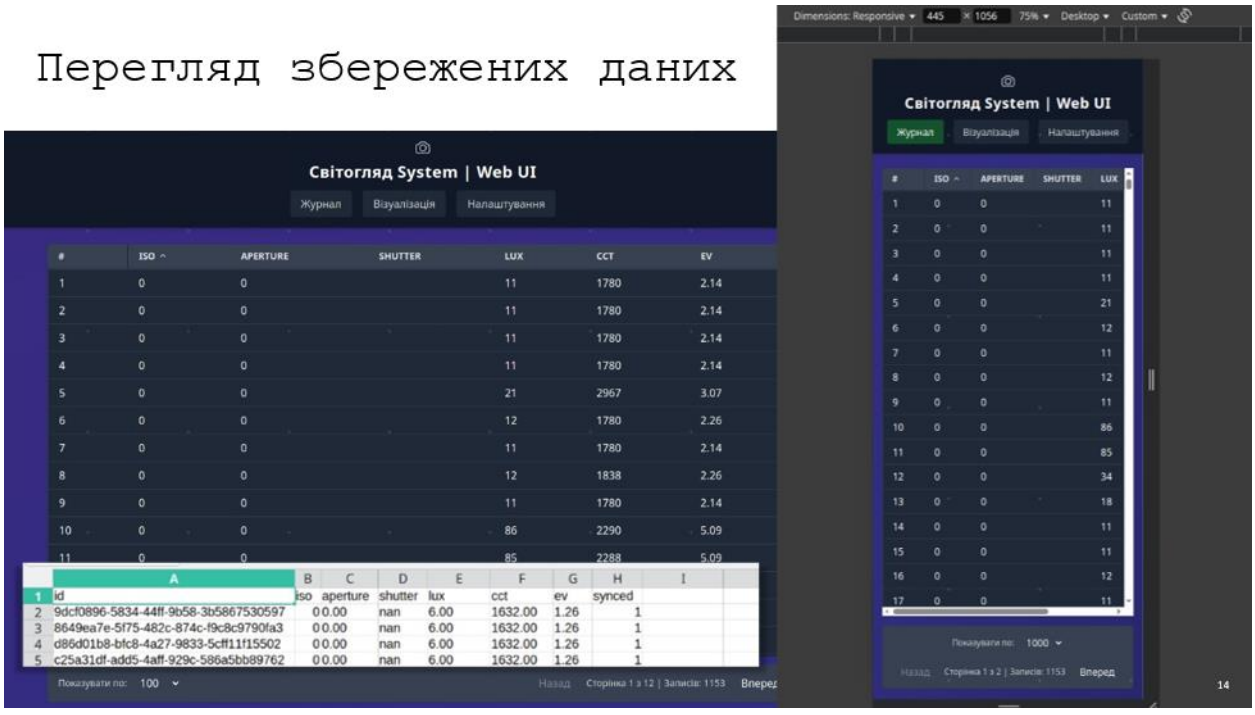
12

Розгортання та мережева взаємодія



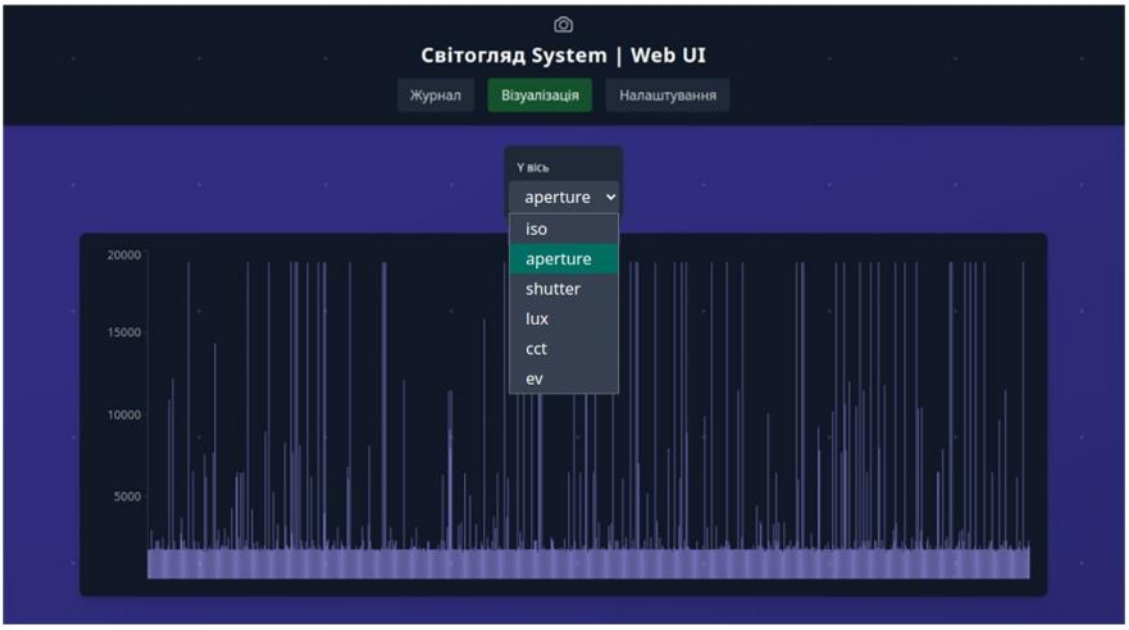
13

Перегляд збережених даних



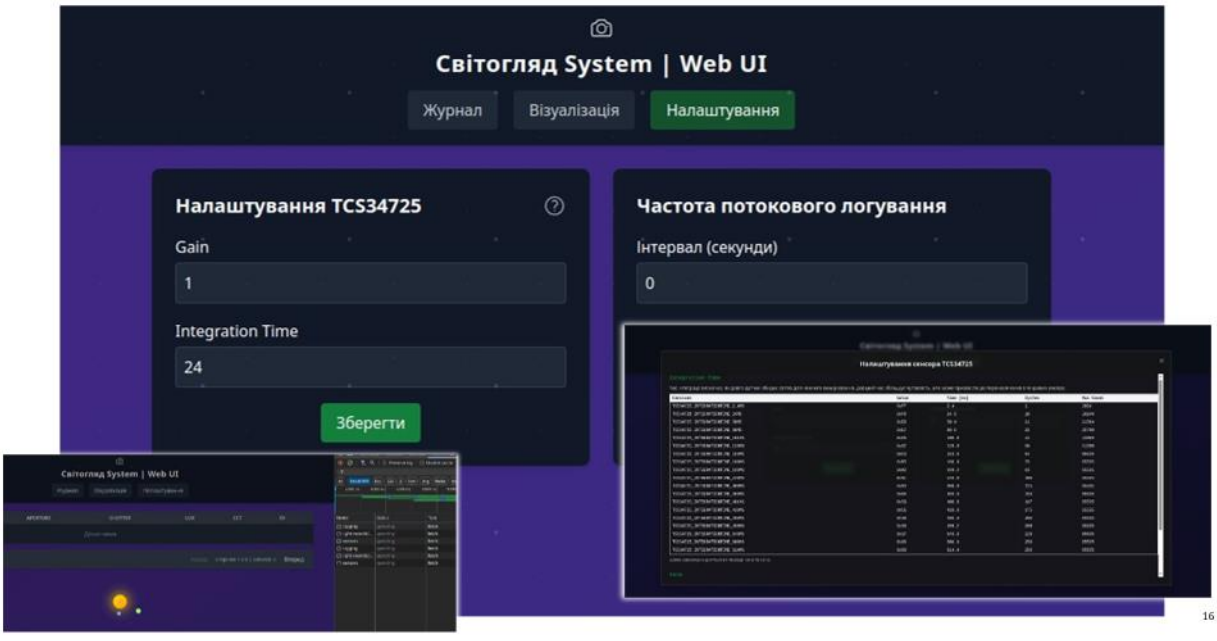
14

Візуалізація збережених даних



15

Налаштування пристрою через WEB UI



16

Висновки

З використанням платформ Arduino та Лілка реалізована система логування та аналізу умов зйомки для плівкової фотографії.

Результати роботи були представлені на виставці технічної творчості на 29-му Міжнародному молодіжному форумі **«Радіоелектроніка та молодь у XXI столітті»** (експонат отримав нагороду за 2 місце), а також опубліковані тези на XV міжнародної науково-технічної конференції **«Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»**.

ДОДАТОК Б

Вихідний код програмних засобів

Б.1 Прототип на Arduino

Б.1.1 Клас кнопки

```
#ifndef BUTTON_H
#define BUTTON_H
#include <Arduino.h>
class Button {
    private:
        uint8_t pin;
        bool state;
        unsigned long debounceDelay;
        unsigned long lastDebounceDelay;
    public:
        Button(uint8_t pin, unsigned long debounceDelay = 10);
        void begin();
        bool isPressed();
        bool wasPressed();
        bool wasReleased();
};
#endif
```

Б.1.2 Клас дисплею

```
#ifndef DISPLAY_H
#define DISPLAY_H
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ui/UserInterface.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define I2C_ADDRESS 0x3C
const char isoText[] PROGMEM = "ISO ";
const char fText[] PROGMEM = "F";
const char shText[] PROGMEM = "1";
const char cmText[] PROGMEM = "cm  ";
const char evText[] PROGMEM = " EV";
const char cctText[] PROGMEM = " K";
```

```
class Display : protected Adafruit_SSD1306 {
    public:
        Display();
        void begin();
        void draw(FL_Parameter marked, FL_Parameter selected,
int iso, float aperture, int shutter, int distance, float ev,
float cct);
    private:
        void drawBoldText(const __FlashStringHelper* text, int
x, int y, int bold);
        void drawUnderlinedText(const __FlashStringHelper* text,
int x, int y);
};
#endif
```

Б.1.3 Клас світлового сенсору

```
#ifndef LIGHT_SENSOR_H
#define LIGHT_SENSOR_H
#include <Wire.h>
#include <Arduino.h>
#include "Adafruit_TCS34725.h"
class LightSensor : protected Adafruit_TCS34725 {
    public:
        LightSensor();
        void begin();
        float getEV();
        float getCCT();
        float calculateShutter(float &ev, int &iso, float
&aperture);
        void printToSerial();
};
#endif
```

Б.1.4 Клас ультразвукового сенсору

```
#ifndef ULTRASONIC_H
#define ULTRASONIC_H
#include <Arduino.h>
class Ultrasonic {
    public:
        Ultrasonic(int trigPin, int echoPin, int
measurementsCount);
        void begin();
        long getDistanceCM();
        long getFilteredDistanceCM();
    private:
        int trigPin;
```

```
        int echoPin;
        int measurementsCount;
};
#endif
```

Б.1.5 Клас інтерфейсу користувача

```
#ifndef USER_INTERFACE_H
#define USER_INTERFACE_H

enum FL_Parameter {
    FL_NONE = 0,
    FL_ISO = 1,
    FL_APERTURE = 2,
    FL_SHUTTER = 3,
    FL_FILM = 4,
};

class UserInterface {
    private:
        int ISO_VALUES[16] = {0, 15, 50, 100, 200, 250, 400,
500, 600, 800, 1000, 1200, 1600, 1800, 3200, 6400};
        float APERTURE_VALUES[12] = {0, 1.4, 2, 2.8, 3.5, 4,
5.6, 8, 11, 16, 22, 64};
        int SHUTTER_VALUES[12] = {0, 1000, 500, 250, 125, 60,
30, 15, 8, 4, 2, 1};
        int ISO_SIZEOF = 16;
        int APERTURE_SIZEOF = 12;
        int SHUTTER_SIZEOF = 12;
        FL_Parameter marked;
        FL_Parameter selected;
        int isoIndex;
        int apertureIndex;
        int shutterIndex;
        int filmIndex;
        int getNextIndex(int index, int maxIndex, bool buttonUp,
bool buttonDown);
    public:
        UserInterface();
        void handleUI(bool buttonUp, bool buttonDown, bool
buttonAccept);

        FL_Parameter getMarked();
        FL_Parameter getSelected();
        int getISO();
        float getAperture();
        int getShutter();
};
#endif
```

Б.1.6 Допоміжний клас для сканування сенсорів

```
#include <./tools/I2C_Scanner.h>
void I2C_Scanner::scan() {
  Wire.begin();
  Serial.println("Scanning I2C bus...");
  for (byte addr = 1; addr < 127; addr++) {
    Wire.beginTransmission(addr);
    if (Wire.endTransmission() == 0) {
      Serial.print("Found device at 0x");
      Serial.println(addr, HEX);
    }
  }
}
```

Б.2 Прототип на Лілка

Б.2.1 Клас розділу світлометрії

```
#ifndef LIGHT_METER_H
#define LIGHT_METER_H
#include <ui/menu-component/MenuComponent.h>
class LightMeter : protected MenuComponent {
    public:
        LightMeter();
        void drawUI(lilka::Canvas *canvas, float lux, float
cct);
    private:
        void drawGradientLuxBar(lilka::Canvas *canvas);
        void drawGradientKelvinBar(lilka::Canvas *canvas);
        void drawCrosshair(lilka::Canvas *canvas, int luxY, int
kelX, int lux, int kelvin);
        int mapLuxToY(float lux);
        int mapKelvinToX(float kelvin);
};
#endif
```

Б.2.2 Клас розділу експозиції

```
#ifndef EXPOSURE_H
#define EXPOSURE_H
#include <ui/menu-component/MenuComponent.h>
enum class FL_Parameter {
    NONE,
    ISO,
```

```
    APERTURE,
};
class Exposure : protected MenuComponent {
    public:
        Exposure();
        void drawUI(lilka::Canvas *canvas, float currentEV,
float recommendedEvMin, float recommendedEvMax, float shutter);
        void handleParameters(lilka::State *state);
        int getISO();
        float getAperture();
        bool evDifferenceOK(float current, float min, float
max);
        bool evDifferenceWARN(float current, float min, float
max);
        bool evDifferenceCRIT(float current, float min, float
max);
    private:
        int ISO_VALUES[16] = {0, 15, 50, 100, 200, 250, 400,
500, 600, 800, 1000, 1200, 1600, 1800, 3200, 6400};
        float APERTURE_VALUES[12] = {0, 1.4, 2, 2.8, 3.5, 4,
5.6, 8, 11, 16, 22, 64};
        int ISO_SIZEOF = 16;
        int APERTURE_SIZEOF = 12;
        FL_Parameter selected;
        int isoIndex;
        int apertureIndex;
        int getNextIndex(int index, int maxIndex, bool buttonUp,
bool buttonDown);
};
#endif
```

Б.2.3 Клас головного меню

```
#ifndef FL_MENU_H
#define FL_MENU_H
#include <lilka.h>
enum class FL_Menu_Enum {
    LIGHT_METER,
    EXPOSURE,
    WIFI_CONFIG,
};
class FL_Menu : private lilka::Menu {
    public:
        FL_Menu();
        void begin(lilka::Controller *controller);
        void drawMenu(lilka::Canvas *canvas);
        bool isSelected();
        void setSelected(bool state);
        bool isLightMeter();
        bool isExposure();
        bool isWiFiConfig();
```

```
    private:
        lilka::Canvas *canvas;
        bool selected;
};
#endif
```

Б.2.4 Базовий клас меню

```
#include <ui/menu-component/MenuComponent.h>
MenuComponent::MenuComponent() {}
void MenuComponent::drawCommonUI(lilka::Canvas *canvas) {
    canvas->fillScreen(lilka::colors::Black); // Clear display
    // Exit info
    canvas->setCursor(60, 210);
    canvas->setTextColor(lilka::colors::Graygrey);
    canvas->print(String("Start для виходу"));
}
```

Б.2.5 Клас функціоналу логування

```
#ifndef LOGGER_H
#define LOGGER_H
#include <lilka.h>
#include <SD.h>
#include <ArduinoJson.h>
enum class LoggerMode {
    SUSPENDED,
    SINGLE,
    STREAM,
};
class Logger {
    public:
        Logger();
        void begin();
        void handleLogging(lilka::State *state, int &iso, float
&aperture, float &shutter, float &lux, float &cct, float &ev);
        JsonDocument readRecords(size_t limit);
        bool markAsSynced(size_t limit);
        size_t countUnsyncedRecords();
        bool applySettings(JsonDocument settings);

    private:
        String fileName = "/fl_logs.csv";
        String dataSeparator = F(",");
        const char* columnNames[8] = {"id", "iso", "aperture",
"shutter", "lux", "cct", "ev", "synced"};
        size_t columns = sizeof(columnNames) /
sizeof(columnNames[0]);
```

```
        LoggerMode mode = LoggerMode::SUSPENDED;
        int streamIntervalSec = 0;
        void pauseStream();
        bool synced(String &columnName, String &value);

        String getHeader();
        String getColumnName(size_t index);
        String generateUUIDv4();
        void saveData(int &iso, float &aperture, float &shutter,
float &lux, float &cct, float &ev);
};
#endif
```

Б.2.6 Клас роботи з мережею

```
#ifndef NETWORK_SERVICE_H
#define NETWORK_SERVICE_H
#include <string>
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <.secrets.h>

class NetworkService {
    public:
        NetworkService();
        JsonDocument get(String suburl);
        JsonDocument post(String suburl, JsonDocument &data);
        JsonDocument patch(String suburl, JsonDocument &data);
        String encodeURL(String str);
        String getMessage(JsonDocument& doc);
        JsonDocument getData(JsonDocument& doc);

    private:
        String getURL(String suburl);
        JsonDocument jsonify(String& response);
};
#endif
```

Б.2.7 Клас взаємодії з бекендом

```
#ifndef API_SERVICE_H
#define API_SERVICE_H
#include <services/network-service/NetworkService.h>
class APIService {
    public:
        APIService();
        String checkConnection();
```

```
        String exportRecords(JsonDocument &records);
        JsonDocument getTSC34725Settings();
        JsonDocument getLoggingSettings();
    private:
        NetworkService networkService;
        String handleStringApiResponse(JsonDocument &response);
};
#endif
```

Б.2.8 Клас налаштувань мережі та синхронізації

```
#include <ui/wifi-config/WiFiConfig.h>
WiFiConfig::WiFiConfig() {}
void WiFiConfig::begin() {
    Serial.println("Connect to network...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}
String WiFiConfig::syncData() {
    APIService apiService;
    Logger logger;
    size_t limit = 10;
    JsonDocument records = logger.readRecords(limit);
    if (records.is<JsonArray>() &&
!records.as<JsonArray>().isNull() &&
records.as<JsonArray>().size() > 0) {
        String result = apiService.exportRecords(records);
        if (result.length() != 0) {
            size_t rows = records.as<JsonArray>().size();
            bool updated = logger.markAsSynced(rows);
            size_t unsyncedRows = logger.countUnsyncedRecords();
            if (updated) {
                return String(F("Успішно збережених \nрядків:
")) + rows + String(F(", залишилось: ")) + unsyncedRows;
            }
            return F("Синхронізація не заве-\nршена.");
        }
        return F("Помилка при збереженні даних.");
    } else {
        return F("Даних для синхроні-\nзації не знайдено.");
    }
}

String WiFiConfig::updateHardware(Logger &logger) {
    APIService apiService;
    LightSensor lightSensor;
    JsonDocument sensorSettings =
apiService.getTSC34725Settings();
    JsonDocument loggingSettings =
apiService.getLoggingSettings();
    if (!sensorSettings.isNull() && !loggingSettings.isNull()) {
        bool sensorUpdated =
```

```
lightSensor.applySettings(sensorSettings);
        bool loggingUpdated =
logger.applySettings(loggingSettings);
        if (sensorUpdated && loggingUpdated) {
            return F("Налаштування оновлено.");
        } else if (!sensorUpdated && !loggingUpdated) {
            return F("Помилка оновлення всіх налаштувань.");
        } else if (!sensorUpdated) {
            return F("Помилка оновлення сенсора.");
        } else {
            return F("Помилка оновлення логування.");
        }
    } else {
        return F("Не вдалося отримати налаштування.");
    }
}
String WiFiConfig::initConnection(Logger &logger) {
    APIService apiService;
    String result = apiService.checkConnection();
    if (result.length() != 0) {
        result = updateHardware(logger);
        if (result.length() != 0) {
            connectionChecked = true;
            return result;
        }
    }
    return F("Сервер не відповідає");
}
void WiFiConfig::drawUI(lilka::Canvas *canvas, Logger &logger) {
    drawCommonUI(canvas);
    lilka::ProgressDialog progress(F("   WiFi-мережа"),
F("Зачекайте, йде підклю-\nчення до мережі WiFi..."));
    int attempts = 0;
    while ((WiFi.status() != WL_CONNECTED) && (attempts <
maxAttempts)) {
        connectionChecked = false;
        begin();
        progress.setProgress(attempts);
        progress.draw(canvas);
        lilka::display.drawCanvas(canvas);
        delay(100);
        attempts++;
    }
    if (WiFi.status() == WL_CONNECTED) {
        String result = "";
        if (!connectionChecked) {
            result = initConnection(logger);
        } else {
            // TODO :: переробити з прогрес баром,
            // щоб показувати скільки залишилось (замість
тексту).
            // Завантаження повинно відбуватися без затримок.
            // При відкриванні цього меню спочатку показувати
```

```
UI, а потім робити
            // мережні запити, бо підлагує.
            result = syncData();
        }
        progress.setProgress(maxAttempts);
        progress.setMessage(result);
    } else {
        progress.setMessage(F("Помилка підключення :("));
    }
    progress.draw(canvas);
    lilka::display.drawCanvas(canvas);

    // TODO :: використати millis,
    // тоді затримувати сповіщення, щоб користувач встигав
прочитати
    delay(3000);
}
```

Б.3 Серверна частина

Б.3.1 Схема Prisma ORM

```
datasource db {
  url      = env("DATABASE_URL")
  provider = "postgresql"
}
generator client {
  provider = "prisma-client-js"
}
model LightRecord {
  id        String   @id @default(uuid()) @db.Uuid
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  iso       Int
  aperture  Float
  shutter   Float?
  lux       Float
  cct       Float
  ev        Float
}
model Sensor {
  id              String   @id @default(uuid()) @db.Uuid
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt
  type            Int      @db.SmallInt @unique
  gain            Int
  integrationTime Int
}
```

```
model Logging {
  id                String   @id @default(uuid()) @db.Uuid
  createdAt         DateTime @default(now())
  updatedAt         DateTime @updatedAt
  streamIntervalSec Int
}
```

Б.3.2 Інтерфейси світлометрії

```
import { ISimpleDTO } from "./simple.dto";
export interface INewLightRecordDTO {
    id: string;
    iso: string;
    aperture: string;
    shutter: string;
    lux: string;
    cct: string;
    ev: string;
    synced: string;
}
export interface ILightRecordDTO extends ISimpleDTO {
    iso: number;
    aperture: number;
    shutter: number;
    lux: number;
    cct: number;
    ev: number;
}
```

Б.3.3 Інтерфейс сенсорів

```
import { SensorType } from "../enums/sensor-type.enum";
import { ISimpleDTO } from "./simple.dto";

export interface ISensorDTO extends ISimpleDTO {
    type: SensorType;
    gain: number;
    integrationTime: number;
}
```

Б.3.4 Інтерфейс налаштувань логування

```
import { ISimpleDTO } from "./simple.dto";
export interface ILoggingDTO extends ISimpleDTO {
    streamIntervalSec: number;
}
```

Б.3.5 Скрипт ініціалізації бази даних

```
import { PrismaClient } from '@prisma/client';
import { SensorType } from '../src/domain/enums/sensor-
type.enum';
const prisma = new PrismaClient();
async function main() {
  await prisma.sensor.create({
    data: {
        type: SensorType.TCS34725,
        gain: 0,
        integrationTime: 214,
    },
  });
  await prisma.logging.create({
    data: {
        streamIntervalSec: 1,
    },
  });
}

main()
  .catch((e) => {
    console.error(e);
    process.exit(1);
  })
  .finally(async () => {
    await prisma.$disconnect();
  });
```

Б.3.6 Клас конфігурації бекенду

```
const DEFAULT_PORT = 5000;
const DEFAULT_TOKEN_EXPIRES_HOURS = 5;
const DEFAULT_JSON_LIMIT_MB = 10;
enum Environment {
    DEV = 'development',
    PROD = 'production',
    TEST = 'testing'
}
export class Config {
    static app = {
        PORT: DEFAULT_PORT,
        JSON_LIMIT_MB: DEFAULT_JSON_LIMIT_MB,
        JWT_SECRET_KEY: undefined,
        NODE_ENV: Environment.DEV,
        TOKEN_EXPIRES_HOURS: DEFAULT_TOKEN_EXPIRES_HOURS
    }
    static db = {
```

```
        DATABASE_URL: undefined
    }
    static get isProduction() {
        return Config.app.NODE_ENV === Environment.PROD;
    }
    static setVariables() {
        Config.app = {
            PORT: parseInt(process.env['PORT']),
            JSON_LIMIT_MB:
parseInt(process.env['JSON_LIMIT_MB']),
            JWT_SECRET_KEY: process.env['JWT_SECRET_KEY'],
            NODE_ENV: process.env['NODE_ENV'] as Environment,
            TOKEN_EXPIRES_HOURS:
parseInt(process.env['TOKEN_EXPIRES_HOURS'])
        }
        Config.db = {
            DATABASE_URL: process.env['DATABASE_URL']
        }
        if (!Config.app.JWT_SECRET_KEY) {
            throw new Error("JWT_SECRET_KEY not provided.");
        }
        if (!Config.db.DATABASE_URL) {
            throw new Error("DATABASE_URL not provided.");
        }
    }
}
```

Б.3.7 Точка входу серверного застосунку

```
import express from "express";
import { Config } from "./config";
import { LightRecordRouter } from "./routes/light-
record.routes";
import { SensorRouter } from "./routes/sensor.routes";
import { LoggingRouter } from "./routes/logging.routes";
import { SystemRouter } from "./routes/system.routes";
const cors = require("cors");
const API_V1 = "/api/v1";
Config.setVariables();
const app = express();
app.use(cors());
app.use(express.json({ limit: `${Config.app.JSON_LIMIT_MB}mb`
}));
// Public routes
app.use(`${API_V1}/system`, SystemRouter);
app.use(`${API_V1}/light-records`, LightRecordRouter);
app.use(`${API_V1}/sensors`, SensorRouter);
app.use(`${API_V1}/logging`, LoggingRouter);

app.listen(Config.app.PORT, () => {
  try {
```

```
    return console.log(`server is listening on
${Config.app.PORT}`);
  } catch {
    return console.error(`server error (on
${Config.app.PORT})`);
  }
});
```

Б.3.8 Базовий абстрактний клас сервісів

```
import { PrismaClient } from "@prisma/client";
import { Prisma } from "../database/prisma-instance";
import { IOffsetPagination } from
"../domain/tools/service.type";
export abstract class SimpleService {
    protected _dbInstance: PrismaClient;
    protected static ALL_PAGES = -1;

    constructor(dbInstance: PrismaClient) {
        this._dbInstance = dbInstance || Prisma.instance;
    }

    protected calculateOffset(page: number, limit: number):
IOffsetPagination {
        limit = Math.abs(limit);
        // const isAllPages = page ===
PaginationService.ALL_PAGES
        return {
            take: limit,
            skip: limit * (Math.abs(page) - 1)
        }
    }
    protected calculateTotalPage(totalRows: number, limit:
number): number {
        if (totalRows <= 0 || limit <= 0) {
            return 0;
        }
        return Math.ceil(totalRows / limit);
    }
}
```

Б.3.9 Базовий абстрактний клас мапперів

```
import { IMapper } from "../domain/tools/mapper.type";

export abstract class SimpleMapper<Model, DTO> implements
IMapper<Model, DTO> {
    // protected _fromDTOFields: string[] = [];
```

```
    protected _toDTOFields: string[] = [];

    // fromDTO(dto: DTO): Model {
    //     return this._fromDTOFields.reduce((model: Model,
field: string) => {
    //         model[field] = dto[field];
    //         return model;
    //     }, {} as Model);
    // }

    toDTO(model: Model): DTO {
        return this._toDTOFields.reduce((dto: DTO, field:
string) => {
            dto[field] = model[field];
            return dto;
        }, {} as DTO);
    }
}
```

Б.3.10 Функції валідації параметрів

```
import { NextFunction, Request, Response } from "express";
import { StatusCodes } from "http-status-codes";
import { ObjectSchema } from "joi";
import { ApiRequest } from "../handlers/request.handler";
import { sendResponse } from "../handlers/response.handler";
import { parse } from "csv-parse";

export const validateParams = (schema: ObjectSchema) => {
  return (req: Request, res: Response, next: NextFunction) => {
    const params = { ...req.body, ...req.query, ...req.params }
    const { error } = schema.validate(params);

    if (!error) {
      ApiRequest.setValidatedParams(req, params);
      next();
    }
    if (!error) {
      return;
    }
    if (!error.details[0].context.key) {
      return sendResponse(res, StatusCodes.BAD_REQUEST, "An
error occurred.");
    }
    return sendResponse(res, StatusCodes.BAD_REQUEST,
error.details[0].message);
  };
};

export const validateCSV = (schema: ObjectSchema, fileField =
"file") => {
```

```
  return (req: Request, res: Response, next: NextFunction) => {
    const file = req[fileField];

    if (!file || !file.buffer) {
      return sendResponse(res, StatusCodes.BAD_REQUEST, "CSV
file not provided.");
    }

    const rows: any[] = [];
    const parser = parse(file.buffer.toString(), {
      columns: true,
      skip_empty_lines: true,
      trim: true,
    });

    parser.on("readable", () => {
      let row;

      while ((row = parser.read()) !== null) {
        const { error } = schema.validate(row);
        if (error)
        {
          return parser.destroy(error);
        }
        rows.push(row);
      }
    });

    parser.on("error", (err) => {
      return sendResponse(res, StatusCodes.BAD_REQUEST, `CSV
validation error: ${err.message}`);
    });
    parser.on("end", () => {
      ApiRequest.setValidatedCSV(req, rows);
      next();
    });
  };
};
```

Б.4 Клієнтський застосунок

Б.4.1 Точка входу в застосунок

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App.tsx";
import "./index.css";
```

```
ReactDOM.createRoot(document.getElementById("root")!).render(
    <React.StrictMode>
        <BrowserRouter>
            <App />
        </BrowserRouter>
    </React.StrictMode>,
);
```

Б.4.2 Головний компонент застосунку

```
import { Route, Routes } from "react-router-dom";
import LogsPage from "./pages/logs-page/LogsPage";
import ChartsPage from "./pages/charts-page/ChartsPage";
import SettingsPage from "./pages/settings-page/SettingsPage";
import { useLightRecordsStore } from
"./store/useLightRecordsStore";
import { useEffect } from "react";
import { useSettingsStore } from "./store/useSettingsStore";
import Loader from "./shared/Loader";
import { useAppStore } from "./store/useAppStore";
import NotFound from "./pages/NotFound";

const App = () => {
    const { page, limit, fetchRecords } =
useLightRecordsStore();
    const { fetchSettings } = useSettingsStore();
    const { isLoading } = useAppStore();

    useEffect(() => {
        fetchSettings();
        fetchRecords();
    }, [page, limit]);

    return (
        <>
            {isLoading && <Loader />}
            <Routes>
                <Route path="/" element={<LogsPage />} />
                <Route path="/logs" element={<LogsPage />} />
                <Route path="/charts" element={<ChartsPage />}
/>
                <Route path="/settings" element={<SettingsPage
/>} />
                <Route path="*" element={<NotFound />} />
            </Routes>
        </>
    );
};

export default App;
```

Б.4.3 Локальне сховище даних світлометрії

```
import { create } from 'zustand'
import { LightRecord } from '../domain/models/LightRecord';
import { lightRecordServiceFactory } from '../services/light-
record-service';
import { PaginateModel } from '../domain/tools/PaginateModel';
import { withLoading } from './useAppStore';
type LightRecordsState = {
    page: number;
    limit: number;
    records: PaginateModel<LightRecord>;
    resetPage: () => void;
    setPrevPage: () => void;
    setNextPage: () => void;
    setLimit: (limit: number) => void;
    fetchRecords: () => Promise<void>;}
const lightRecordService = lightRecordServiceFactory();
export const useLightRecordsStore =
create<LightRecordsState>((set, get, state) => ({
    page: lightRecordService.DEFAULT_RECORDS_PAGE,
    limit: lightRecordService.DEFAULT_RECORDS_LIMIT,
    records: lightRecordService.DEFAULT_RECORDS_OBJ,
    resetPage: () => set({ page:
lightRecordService.DEFAULT_RECORDS_PAGE }),
    setPrevPage: () => set({ page: Math.max(1,
state.getState().page - 1) }),
    setNextPage: () => set({ page:
Math.min(state.getState().records.totalPage,
state.getState().page + 1) }),
    setLimit: (limit) => set({ limit }),

    fetchRecords: withLoading(async () => {
        const { page, limit } = get();
        const records = await
lightRecordService.getRecords(page, limit);
        set({ records });
    }),
}));
```

Б.4.4 Стан застосунку

```
import { create } from 'zustand'

type AppState = {
    isLoading: boolean
    setLoading: (value: boolean) => void
}
export const useAppStore = create<AppState>((set) => ({
```

```
    isLoading: false,
    setLoading: (value: boolean) => set({ isLoading: value }),
}))
export function withLoading<T extends (...args: any[]) =>
Promise<any>>(fn: T): T {
  return (
async (...args: Parameters<T>): Promise<ReturnType<T>> => {
    const { setLoading } = useAppStore.getState();
    setLoading(true)
    try {
      return await fn(...args)
    } finally {
      setLoading(false)
    }
  }) as T;
}
```

Б.4.5 Базовий абстрактний клас сервісів

```
import ky from "ky";
export abstract class ApiService {
    protected API = ky.create({
        prefixUrl: import.meta.env.VITE_API_URL,
    }).extend({
        headers: {
            "ngrok-skip-browser-warning":
import.meta.env.VITE_SKIP_NGROK_WARNING,
        },
    });
}
```

Б.4.6 Базовий компонент модального вікна

```
import { FC, ReactNode, useEffect } from "react";
import { XMarkIcon } from "@heroicons/react/24/outline";
interface ModalWindowProps {
    opened: boolean;
    setOpened: (value: boolean) => void;
    title: string;
    body: ReactNode;
}
const ModalWindow: FC<ModalWindowProps> = (props) => {
    useEffect(() => {
        if (!props.opened) return;
        const handleKeyDown = (e: KeyboardEvent) => {
            if (e.key === "Escape") {
                props.setOpened(false);
            }
```

```
        };
        window.addEventListener("keydown", handleKeyDown);
        return () => window.removeEventListener("keydown",
handleKeyDown);
    }, [props.opened, props.setOpened]);
    if (!props.opened) {
        return <></>;
    }
    return (
        <div className="fixed inset-0 z-50 flex items-center
justify-center bg-black bg-opacity-40 backdrop-blur-sm">
            <div className="bg-neutral-900 bg-opacity-80
rounded-none shadow-2xl w-[80vw] h-[80vh] p-6 relative border
border-neutral-700 flex flex-col">
            <button
                className="absolute top-2 right-2 text-gray-400
hover:text-gray-200 transition-colors text-2xl"
                onClick={() => props.setOpened(false)}
                aria-label="Закрити"
            >
                <XMarkIcon className="w-6 h-6 mt-2 mr-2 text-
gray-400" />
            </button>
            <h2 className="text-xl font-semibold mb-4 text-gray-
100 text-center">{props.title}</h2>
            <div className="flex-1 overflow-auto">
                {props.body}
            </div>
            </div>
        </div>
    );
};
export default ModalWindow;
```