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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Комп'ютеризована система збору та логування показників сенсорів для пожежної сигналізації

Виконав(ла): студент(ка) *IV* курсу, групи *CI-41*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

		Гаврада Д.М.
	(підпис)	(прізвище та ініціали)
Керівник		Яцишин В.В.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Тиш Є.В.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Осухівська Г.М.
	(підпис)	(прізвище та ініціали)
Рецензент		Карпінський М.П.
	(підпис)	(прізвище та ініціали)

Тернопіль
2024

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

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

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
Осухівська Г.М.
(підпис) (прізвище та ініціали)
« 25 » 04 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

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

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

студенту Гаврада Дмитро Миколайович
(прізвище, ім'я, по батькові)

1. Тема роботи Комп'ютеризована система збору та логування показників сенсорів для пожежної сигналізації

Керівник роботи Яцишин Василь Володимирович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *24* » *04* *2024* року № *4/7-408*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Технічне завдання

4. Зміст роботи (перелік питань, які потрібно розробити
Аналіз вимог та особливостей побудови системи
Проектування апаратного забезпечення системи
Реалізація програмного забезпечення та тестування системи
Безпека життєдіяльності, основа охорони праці

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Структурна схема приладу
Структурна схема веб-інтерфейсу
Схема електрично-принципова
Блок-схема алгоритму

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.04.24

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Отримання індивідуального завдання	01.02 – 09.02	Виконано
2.	Аналіз отриманого завдання	05.02 – 11.02	Виконано
3.	Ознайомлення з документацією мікроконтролера ESP8266	27.04.24 – 28.04.24	Виконано
4.	Ознайомлення з документацією сенсора MQ2	29.04 – 30.04	Виконано
5.	Ознайомлення з документацією сенсора DHT11	01.05 – 02.05	Виконано
6.	Ознайомлення з бібліотеками створення WebServer на мікроконтролері ESP8266	03.05 – 04.05	Виконано
7.	Ознайомлення з бібліотеками перетворення даних з датчиків для передавання в базу даних та на створений WebServer на мікроконтролері ESP8266	05.05.24 – 06.05.24	Виконано
8.	Створення електрично-принципової схеми пристрою	07.05 – 08.05	Виконано
9.	Проектування пристрою за схемою	09.05 – 10.05	Виконано
10.	Написання програмного забезпечення	13.05 – 19.05	Виконано
11.	Тестування програмного забезпечення	20.05.24	Виконано
12.	Тестування створеного пристрою	21.05.24	Виконано
13.	Безпека життєдіяльностіі, основи охорони праці	10.06 – 15.06	Виконано
14.	Оформлення кваліфікаційної роботи	16.06 – 20.06	Виконано
15.	Попередній захист кваліфікаційної роботи	14.06	
16.	Захист кваліфікаційної роботи	24.06	

Студент _____ Гаврада Д.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Яцишин В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Комп’ютеризована система збору та логування показників сенсорів для пожежної системи // Гаврада Дмитро Миколайович // Кваліфікаційна робота бакалавра // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних систем та мереж, група СІ-41 // Тернопіль, 2024 // с. – 98, рис. – 44, табл. – 1, аркушів А1 – 4, додат. – 4, бібліогр. – 26.

Ключові слова: комп’ютеризована система, пожежна сигналізація, сенсори, логування даних, веб-сторінка.

Кваліфікаційна робота бакалавра складається з чотирьох розділів.

В першому розділі розглядається аналіз та методи вирішення поставленого завдання. Приводяться в приклад готові системи, їх характеристики, переваги та недоліки.

В другому розділі розглядається структурна схема системи. Обґрунтування вибору мікроконтролера та сенсорів, їх характеристики та можливі аналоги.

В третьому розділі розглядається блок-схема алгоритму. Обґрунтування вибору середовища для розробки. Розробка коду для мікроконтролера та розробка веб-інтерфейсу.

В четвертому розділі розглядаються питання з безпеки життєдіяльності та основи охорони праці.

ABSTRACT

A computerized system for collecting and logging fire alarm sensor’s data // Havrada Dmytro // Bachelor's qualification work // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, group CI-41 // Ternopil, 2024 // p. – 98, fig. – 42, tabl. – 1, A1 sheets – 4, append. – 4, bibliogr. – 26.

Keywords: computerized system, fire alarm, sensor, logging data, web-page.

Qualification work is created with four sections.

In the first section, the analysis and methods for solving the given task are discussed. Examples of existing systems are provided, along with their characteristics, advantages, and disadvantages.

In the second section, the structural diagram of the system is examined. The justification for choosing the microcontroller and sensors is provided, including their characteristics and potential alternatives.

In the third section, the flowchart of the algorithm is reviewed. The rationale for selecting the development environment is discussed. The development of the microcontroller code and the web interface is described.

In the fourth section, issues related to life safety and basic occupational safety principles are considered.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

Wi-Fi – Wireless Fidelity (бездротова правдивість відтворення)

SoC – System on a chip (система на кристалі)

IoT – Internet of Things (інтернет речей)

RF – Radio frequency (радіо-хвиля)

SRAM – static random access memory (статична оперативна пам'ять з довільним доступом)

SDK – Software Development Kit (набір засобів розробки)

ESCP – Espressif Systems' Smart Connectivity Platform

OTA – Over the Air (оновлення по повітрю)

IDE – Integrated Development Environment (інтегроване середовище розробки)

IP – Internet Protocol

HTML – HyperText Markup Language (мова гіпертекстової розмітки)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

ВСТУП

У сучасному світі, де автоматизація та інновації проникають у всі сфери життя, все більшого значення набуває питання забезпечення безпеки людини. Пожежі, на жаль, не є рідкістю, і завдають значних збитків як майну, так і людському життю. Тому розробка ефективних систем пожежної сигналізації є надзвичайно актуальною задачею.

Пожежа – це неконтрольоване займання, що відбувається поза спеціально визначеним місцем і завдає матеріальних збитків. Основними причинами виникнення пожеж є недбалість людей у поводженні з вогнем і порушення правил пожежної безпеки.

Система пожежної сигналізації складається з технічних засобів, які призначені для виявлення пожежі, обробки та передачі повідомлень про неї, а також для надсилання команд на активацію автоматичних систем пожежогасіння, протидимного захисту та інших інженерних або технологічних засобів протипожежного захисту.

Традиційні системи пожежної сигналізації мають обмежені можливості збору та логування даних про стан сенсорів. Це ускладнює аналіз причин пожеж, прийняття обґрунтованих рішень щодо покращення роботи таких систем.

Метою цієї кваліфікаційної роботи є розробка комп'ютеризованої системи збору та логування показників сенсорів пожежної сигналізації, яка буде виконувати наступні завдання:

– збирати та надсилати дані на веб-сторінку з сенсорів в режимі реального часу.

– зберігати дані в базі даних для подальшого аналізу.

– візуалізувати дані про стан сенсорів в вигляді графіків.

У цій кваліфікаційній роботі ми будемо розглядати можливі варіанти та методи вирішення цього завдання, включаючи вибір та інтеграцію різних

типів сенсорів пожежної сигналізації, способи збору та обробки даних, а також передачу інформації в режимі реального часу та її подальше зберігання в базі даних.

Буде розглянуто питання про апаратне забезпечення. Обґрунтування вибору саме цих сенсорів та мікроконтролера, можливі альтернативи цим компонентам. Також буде розглянуто характеристики та розпінування мікроконтролера та сенсорів.

Особлива увага буде приділена розробці веб-інтерфейсу, який дозволить користувачам не лише отримувати оперативну інформацію про стан системи, але й мати можливість виключити її для енергоефективного використання батареї. На веб-сторінці буде можливість дивитись на показники сенсорів в режимі реального часу, графіків згідно показників сенсорів та можливості вибору даних згідно дати та часу з бази даних для відображення.

РОЗДІЛ 1 АНАЛІЗ ВИМОГ ТА ОСОБЛИВОСТЕЙ ПОБУДОВИ СИСТЕМИ ЗБОРУ ТА ЛОГУВАННЯ ПОКАЗНИКІВ СЕНСОРІВ ДЛЯ ПОЖЕЖНОЇ СИГНАЛІЗАЦІЇ

1.1 Аналіз технічного завдання

Розробка комп'ютеризованої системи збору та логування показників сенсорів для пожежної сигналізації починається з детального аналізу технічного завдання. Основними вимогами до системи є забезпечення точного і своєчасного збору даних від сенсорів, що моніторять параметри температури, вологості та диму, які можуть свідчити про виникнення пожежі. Система повинна бути здатна в режимі реального часу обробляти отримані дані, зберігати їх у надійному сховищі та забезпечувати швидкий доступ до історії даних [1].

Основні функції, які має виконувати система, включають:

– збір даних з сенсорів, встановлених в систему;

– автоматичне збереження зібраних даних у базі даних;

– забезпечення користувачів інтерфейсом для перегляду та аналізу даних.

Існує безліч різних систем пожежної сигналізації. Кожна система має певний функціонал та характеристики сенсорів. Розглянемо декілька варіантів готових схем [2].

Першим розглянемо систему X-Sense Mini Smoke Alarm (LPNHE649742163). Це система виявлення диму з звуковим сигналом на батарейках. Країна виробник – Китай, захис тампером від злому відсутній, може виявляти тільки дим, може бути встановлений на стіну або стелю, при

виявлені пожежі буде оповіщати тільки звуковим сигналом та живиться від батарейок. Зовнішній вигляд системи зображено на рис. 1.1.

Рисунок 1.1 – Пожежна сигналізація X-Sense Mini

В цієї системи є свої плючи та мінуси. Плюсами є енергоекономність та невеликі розміри. Мінусами цієї системи є ненадійність через можливе розрядження батарейок в системі що спричинить неможливість оповіщення при виникненні пожежі, оповіщення тільки через звуковий сигнал та немає відображення даних в реальному часі або збереження в базу даних.

Розглянемо ще систему RIAKELL. Ця система є більш універсальнішою та енергоефективнішою ніж попередня. Зовнішній вигляд зображено на рис. 1.2.

Рисунок 1.2 – Пожежна сигналізація RIAKELL

Ця система оснащена сучасним димовим детектором з фотоелектричним сенсором, який оперативно виявляє дим від вогню та попереджає про небезпеку звуковим сигналом на 85 дБ. Коли дим потрапляє до димової камери, інтелектуальний чипсет аналізує його три рази на секунду, щоб уникнути помилкових тривог.

Пожежна сигналізація має функцію самодіагностики. Детектор постійно перевіряє свою працездатність і своєчасно повідомляє про помилки або низький заряд батареї. Вбудований літій-іонний акумулятор, разом з компонентами з низьким енергоспоживанням, знижує експлуатаційні витрати та продовжує термін служби до 10 років, забезпечуючи безперервний захист 24/7. Це допомагає заощадити ваші зусилля та кошти в майбутньому [7].

При аналізі технічного завдання також важливо врахувати вимоги до надійності і безперебійної роботи системи, оскільки від її коректної роботи залежить безпека місця в якому вона встановлена та своєчасне реагування на пожежі. Система повинна мати можливість інтеграції з інженерними мережами та системами безпеки, а також підтримувати масштабованість для можливості розширення функціональності та підключення додаткових сенсорів в майбутньому [2].

У процесі розробки системи необхідно також провести тестування її працездатності в різних умовах експлуатації, зокрема при моделюванні реальних сценаріїв пожежі. Це дозволить виявити та усунути потенційні недоліки та забезпечити високу надійність системи.

Таким чином, розробка комп'ютеризованої системи збору та логування показників сенсорів для пожежної сигналізації є складним та багатоетапним процесом, який вимагає врахування широкого спектру технічних та організаційних аспектів для забезпечення ефективної роботи та високого рівня безпеки місцяв якому вона встановлюється [3].

1.2 Мета створення системи

Розробка комп'ютеризованої системи збору та логування показників сенсорів для пожежної сигналізації має на меті забезпечити надійний та ефективний моніторинг потенційних пожежних загроз в режимі реального часу. З огляду на підвищені вимоги до безпеки в сучасних будівлях, як житлових, так і комерційних, необхідно створити систему, яка не тільки оперативно виявляє загрозу пожежі, але й дає змогу проконтролювати ситуацію з показниками в реальному часі або подивитись на попередньо записані дані з бази даних.

Головна мета полягає в підвищенні рівня безпеки за рахунок впровадження системи, яка здатна виявити загрозу на ранніх стадіях і мінімізувати ризик поширення пожежі. Завдяки використанню сучасних сенсорних технологій, система повинна швидко і точно виявляти зміни у навколишньому середовищі, такі як підвищення температури, наявність диму або газів. Це дозволяє оперативно реагувати на потенційні загрози і вживати відповідних заходів, що значно знижує ймовірність великих збитків і підвищує шанси на збереження людських життів.

Сучасні пожежні сигналізації повинні забезпечувати централізоване збирання та обробку даних з великої кількості сенсорів, встановлених у різних частинах будівлі. Розроблена система дозволяє збирати дані з сенсорів в єдину базу даних. Це дає змогу отримувати інформацію про стан місця в якому ця система встановлена в певні часові проміжки [4].

Однією з важливих цілей створення системи є забезпечення її надійності та безперебійної роботи. Це досягається шляхом використання сучасних апаратних та програмних рішень, що дозволяють системі функціонувати навіть у випадку відмови окремих компонентів.

Розробка системи передбачає впровадження механізмів для детального логування всіх подій, пов’язаних з роботою сенсорів та системи в цілому. Це

включає в себе збирання інформації про спрацювання сенсорів, перевірки системи. Таке логування є критично важливим для аналізу ефективності системи, виявлення потенційних проблем та їх усунення. Збережені файли з даними сенсорів можуть бути використані для проведення розслідувань після інцидентів та для забезпечення відповідності нормативним вимогам.

Комп'ютеризована система збору та логування показників сенсорів для пожежної сигналізації може бути інтегрована з іншими системами безпеки, такими як системи відеоспостереження, контролю доступу та оповіщення. Це дозволяє створити комплексну систему безпеки, яка забезпечує всебічний моніторинг та захист місця в якому вона встановлюється. Інтеграція з іншими системами також сприяє більш ефективному реагуванню на надзвичайні ситуації та координації дій різних служб.

Автоматизація процесів збору, обробки та аналізу даних значно знижує витрати на утримання та експлуатацію системи. Завдяки використанню сучасних технологій, система може працювати з мінімальним втручанням людини, що дозволяє зменшити витрати на персонал та підвищити ефективність роботи. Автоматичне виявлення та аналіз потенційних загроз також сприяє зниженню витрат на ліквідацію наслідків пожеж.

Система повинна відповідати основним вимогам та стандартам у сфері пожежної безпеки. Це включає в себе вимоги до точності та надійності сенсорів, процедур логування. Відповідність нормативним вимогам є важливою умовою для забезпечення достовірності та ефективності роботи системи.

Система повинна забезпечувати зручний доступ до інформації для користувачів, відповідальних за пожежну безпеку. Це включає в себе розробку зручного інтерфейсу для моніторингу стану сенсорів в реальному часу а також доступ до історичних даних. Підвищення рівня інформованості користувачів сприяє більш ефективному управлінню системою та швидкому реагуванню на виникаючі загрози [5].

1.3 Основні завдання та функції системи

Основними завданнями та функціями системи є збір даних та подальше зберігання та відображення даних з сенсорів: газу, вологості та температури. Система також забезпечує аналіз отриманих даних, їх обробку та інтерпретацію в вигляді графіків, що дозволяє своєчасно виявляти відхилення від норми та попереджати про можливі небезпеки.

Комп'ютеризована система збору та логування показників сенсорів для пожежної сигналізації виконує ключову функцію – збір даних від різних типів сенсорів, встановлених на об'єкті. Ці сенсори включають сенсори диму, температури та вологості.

Збір даних від сенсорів відбувається у реальному часі. Сенсори можуть працювати у безперервному режимі. Система повинна забезпечувати стабільний зв'язок з сенсорами та мінімізувати затримку у передачі даних для своєчасного реагування на потенційні загрози.

Одним із основних завдань системи є автоматичне збереження зібраних даних у базі даних. Це забезпечує надійне зберігання великого обсягу інформації та доступ до історичних даних для подальшого аналізу. База даних повинна бути оптимізована для роботи з великими обсягами даних та забезпечувати швидкий доступ до необхідної інформації.

При розробці системи важливо врахувати вимоги до цілісності даних, забезпечення захисту від втрати. Резервне копіювання даних є обов’язковим елементом, що дозволяє зберегти інформацію у разі збою системи або інших непередбачуваних ситуацій.

Система повинна надавати користувачам інтуїтивно зрозумілий інтерфейс для перегляду та аналізу зібраних даних. Інтерфейс користувача має бути зручним, адаптивним і доступним на стаціонарних комп'ютерах.

Основні функції інтерфейсу включають:

– відображення даних в реальному часі. Система повинна відображати поточні показники всіх сенсорів у реальному часі. Це дозволяє оперативно реагувати на зміни умов і потенційні загрози;

– історичний аналіз. Користувачі повинні мати можливість переглядати історичні дані за певний період. Це включає функції фільтрації та сортування даних для полегшення аналізу.

1.4 Вимоги до системи

Система повинна підтримувати збір даних з різних типів сенсорів, таких як димові сенсори та температурні сенсори що можуть бути використані для виявлення пожеж. Система повинна виводити в реальному часі для миттєвого виявлення аномалій, що можуть вказувати на пожежу. Система має підтримувати алгоритми для виявлення хибних спрацьовувань та зниження їх кількості, а також можливість адаптації порогових значень для різних сенсорів залежно від умов експлуатації. Всі зібрані дані повинні записуватись до централізованої бази даних з забезпеченням захищеного збереження даних і можливістю резервного копіювання. Веб-інтерфейс повинен забезпечувати віддалений доступ до системи та перегляд даних з сенсорів.

Система повинна бути високонадійною і мати високу стійкість до збоїв з можливістю швидкого відновлення після них.. Висока швидкість обробки даних є обов'язковою для забезпечення роботи в реальному часі, а також низька затримка між отриманням сигналу від сенсора та реакцією системи. Система повинна підтримувати велику кількість сенсорів з можливістю їх легкого додавання, а також розширення обсягу зберігання даних без зниження продуктивності.

Для обробки та зберігання даних потрібен сервер з відповідними технічними характеристиками для забезпечення вимог до продуктивності.

Необхідні сенсори повинні бути сертифіковані для використання в системах пожежної сигналізації, а комунікаційне обладнання повинно забезпечувати стабільний зв'язок між сенсорами та сервером. Система повинна використовувати спеціалізоване програмне забезпечення для збору, обробки та візуалізації даних.

Підтримка системи з боку виробника повинна включати регулярні оновлення та технічну підтримку. Система повинна відповідати стандартам та нормативним актам, що регулюють системи пожежної безпеки. Регулярне тестування системи є обов'язковим для виявлення та усунення можливих недоліків.

Необхідно провести комплексне тестування на етапі розробки та впровадження системи. Система повинна бути протестована на стійкість до збоїв та здатність швидкого відновлення. Випробування повинні показати здатність системи виявляти реальні пожежі та мінімізувати кількість помилкових спрацьовувань [8].

РОЗДІЛ 2 ПРОЕКТУВАННЯ АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ЗБОРУ ТА ЛОГУВАННЯ ПОКАЗНИКІВ СЕНСОРІВ ДЛЯ ПОЖЕЖНОЇ СИГНАЛІЗАЦІЇ

2.1 Розробка структурної схеми комп’ютеризованої системи

Перед початком роботи потрібно розробити структурну схему комп’ютеризованої системи. Основним елементом комп'ютеризованої системи збору даних для пожежної сигналізації є мікроконтролер, який взаємодіє з сенсорами та надсилає дані на вивід даних. На початковому етапі розробки була створена базова структурна схема яка зображена на рис. 2.1. Вона демонструє загальну архітектуру системи.

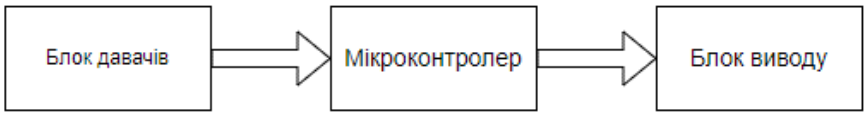

Рисунок 2.1 – Загальна структурна схема системи

На цій схемі представлено три основних компоненти:

– блок давачів(сенсорів) відповідає за збір даних з навколишнього середовища. Цей блок включає сенсори, такі як температури і вологості та наявності диму;

– мікроконтролер – це центральний компонент, який приймає сигнали з сенсорів, обробляє їх та готує для подальшої передачі;

– блок виводу використовується для відображення або зберігання даних. Цей блок включає засоби для відображення даних, такі як веб–сторінка та база даних.

Для більш детального уявлення роботи комп'ютеризованої системи була розроблена розширена структурна схема яка зображена на рис. 2.2. Ця схема включає конкретні компоненти та їх взаємодію.

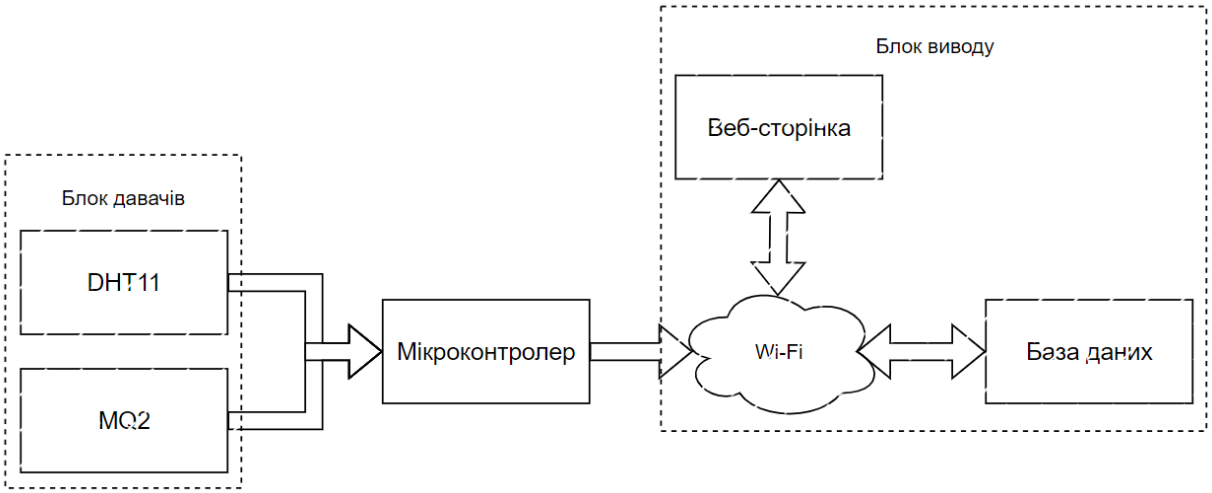

Рисунок 2.2 – Розширена структурна схема системи

На цій схемі представлено наступні елементи:

а) Блок сенсорів включає в собі:

– сенсор DHT11 що вимірює температуру та вологість. Він відправляє цифрові сигнали до мікроконтролера;

– сенсор MQ-2 використовується для вимірювання концентрації диму в повітрі. Він також передає дані цифровими сигналами до мікроконтролера.

б) Мікроконтролер отримує сигнали від сенсорів DHT11 і MQ-2, обробляє їх та передає дані через бездротову мережу Wi-Fi.

в) Wi-Fi модуль забезпечує бездротову передачу даних від мікроконтролера до інших компонентів блоку виводу.

г) Блок виводу включає в собі:

– веб–сторінка дозволяє в режимі реального часу відстежувати показники з сенсорів через інтернет;

– база даних зберігає зібрані дані для можливості подальшого відображення на веб-сторінці.

Таким чином, розроблена структурна схема надає повне уявлення про функціонування комп'ютеризованої системи збору та логування показників сенсорів для пожежної сигналізації. Вона враховує всі необхідні компоненти, від збору даних до їх відображення та зберігання, забезпечуючи ефективну та надійну роботу системи в реальному часі.

Тепер розробимо структурну схему для веб-сайт. Веб-сайту буде складатись з 4 веб-сторінок: головної, сторінки з таблицями які оновлюються в реальному часі, сторінки з графіками, сторінка з таблицею яка підключена до бази даних Розроблену структурну схему зображено на рис. 2.3.

Рисунок 2.3 – Структурна схема веб-сторінки

На головній сторінці повинна знаходитись кнопка включення та виключення. Ще потрібно сторінку з таблицею на якій відображатимуться дані з сенсорів в реальному часі. Сторінка з графіками буде третьою, на ній необхідно створити два поля з графіками які будуть відмальовуватись згідно даних сенсорів. Четвертою буде сторінка з табличкою і можливістю вибору даних за датою та часом. На кожній сторінці мають бути кнопки для переходу на любу іншу сторінку та знизу повинна бути інформація про стан системи.

2.2 Обґрунтування вибору апаратної частини

Комп'ютеризована система збору та логування показників сенсорів для пожежної сигналізації вимагає ретельного підбору апаратної частини для забезпечення надійності, точності та ефективності роботи. Вибір мікроконтролера та сенсорів для такої системи базується на декількох ключових критеріях, включаючи сумісність між собою, продуктивність, енергоспоживання та надійність. Для даного проекту були вибрані мікроконтролер ESP8266, сенсор газу MQ2, сенсор температури та вологості DHT11, а також світлодіоди та резистори для візуальної індикації стану системи. Нижче знаходиться обґрунтування вибору кожного з компонентів та їх порівняння з альтернативними рішеннями.

ESP8266 є популярним мікроконтролером з вбудованим Wi-Fi модулем, що робить його ідеальним вибором для розробки проектів. Він оснащений 32-бітним процесором Tensilica Xtensa LX106, що працює на частоті до 160 МГц. Вбудований Wi-Fi модуль підтримує стандарти 802.11 b/g/n, що забезпечує бездротове підключення. Також ESP8266 має флеш-пам'ять до 4 МБ, високу кількість GPIO (General Purpose Input/Output) пінів і може підтримуваим різноманітні протоколи зв'язку, включаючи SPI, I2C та UART.

Альтернативними варіантами могли б бути мікроконтролери, такі як Arduino Uno або Raspberry Pi Zero W, проте ESP8266 має кілька суттєвих переваг. По-перше, він має вбудований Wi-Fi модуль, що значно спрощує створення бездротових систем. Arduino Uno не має вбудованого Wi-Fi, що вимагало б додаткових модулів. По-друге, ESP8266 є компактним і енергоефективним, що важливо для систем, які працюють на батареях. Також ESP8266 є більш доступним у порівнянні з Raspberry Pi Zero W, що робить його економічно вигідним рішенням для масового виробництва.

MQ2 є популярним сенсором для виявлення газів, таких як пропан, бутан, метан, етанол, водень та дим. Він має високу чутливість до широкого

спектру газів, швидкий час відгуку і простоту інтеграції з мікроконтролерами завдяки аналоговому виходу.

Альтернативними варіантами сенсора газу могли б бути такі як MQ3 (для парів алкоголю) або MQ7 (для моноксиду вуглецю). Проте MQ2 має кілька переваг. Він здатний виявляти широкий спектр газів, що робить його більш універсальним для систем пожежної сигналізації. Також MQ2 є одним з найдоступніших сенсорів газу на ринку, що знижує загальну вартість системи.

DHT11 є популярним сенсором для вимірювання температури та вологості. Він забезпечує діапазон вимірювання температури від 0°C до 50°C з точністю ±2°C і діапазон вимірювання вологості від 20% до 90% з точністю ±5%. DHT11 має низьке енергоспоживання і цифровий вихідний сигнал, що спрощує інтеграцію з мікроконтролерами.

Альтернативними варіантами могли б бути сенсори, такі як DHT22 або BME280. Проте DHT11 є значно дешевшим, ніж DHT22 або BME280, що робить його привабливим для бюджетних проектів, також DHT11 має простий цифровий інтерфейс, що спрощує програмування та інтеграцію в системах.

Світлодіоди використовуються для візуальної індикації стану системи. Вибрані червоний та зелений світлодіоди дозволяють відображати різні стани, включену систему та виключену. Для обмеження струму через світлодіоди використовуються відповідні резистори, що забезпечує їх безпечну експлуатацію.

2.2.1 Мікроконтролер ESP8266

ESP8266 від компанії Espressif – це високоінтегроване Wi-Fi SoC (System on Chip) рішення, яке відповідає сучасним вимогам до енергоефективності, компактності та надійної роботи в області Інтернету речей (IoT).

Завдяки своїм автономним Wi-Fi можливостям, ESP8266 може функціонувати як незалежний пристрій або як додатковий модуль для хосту. Коли ESP8266 використовується як основний контролер, він завантажується

безпосередньо з флеш-пам'яті, що забезпечує швидкий старт системи. Вбудований високошвидкісний кеш оптимізує продуктивність і використання системної пам'яті. Крім того, ESP8266 може виконувати роль Wi-Fi адаптера для будь-якого мікроконтролера, підключеного через SPI/SDIO або UART інтерфейси.

ESP8266 включає в себе такі компоненти, як антени, RF балун, підсилювач потужності, малошумний приймальний підсилювач, фільтри і модулі управління живленням. Завдяки компактному дизайну, пристрій дозволяє зменшити розмір друкованої плати і мінімізує потребу в зовнішніх компонентах.

Окрім Wi-Fi можливостей, ESP8266 оснащений удосконаленим 32-бітним процесором Tensilica серії L106 Diamond і вбудованою оперативною пам'яттю (SRAM). Він може взаємодіяти з зовнішніми сенсорами та іншими пристроями через GPIO порти. Набір засобів розробки програмного забезпечення (SDK) включає приклади коду для різних додатків, що полегшує розробку.

Платформа Smart Connectivity Platform (ESCP) від Espressif Systems забезпечує наступні функції:

– швидке перемикання між режимами сну і активності для підвищення енергоефективності;

– адаптивне налаштування радіочастотного сигналу для зменшення споживання енергії;

– розвинені можливості обробки сигналів;

– технології зниження перешкод і співіснування RF (радіо хвилі), що зменшує взаємний вплив з мобільними пристроями, Bluetooth, DDR, LVDS та LCD дисплеями [9].

Зовнішній вигляд ESP8266 зображено на рис. 2.4.

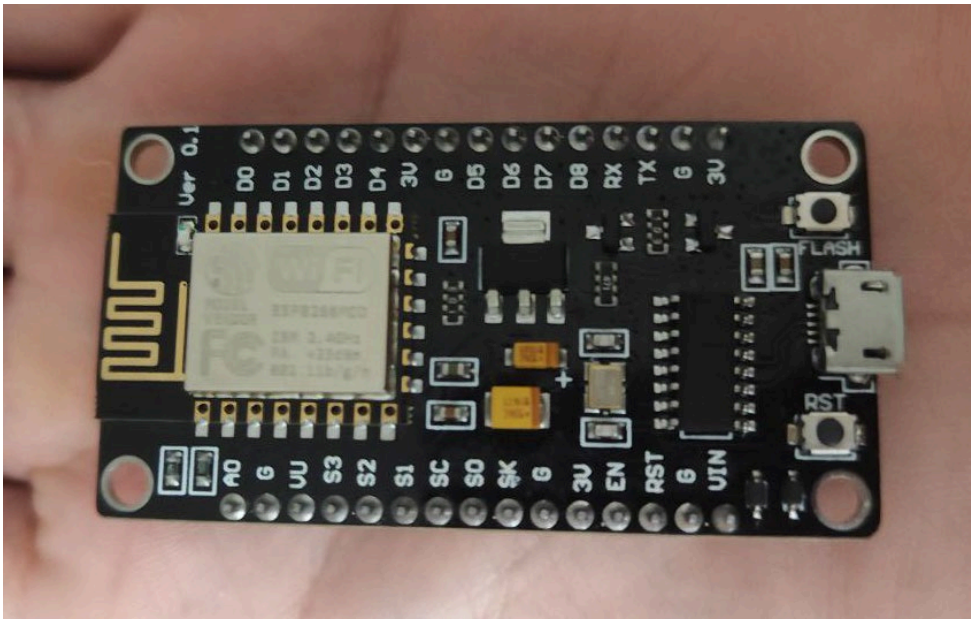

Рисунок 2.4 – Зовнішній вигляд мікроконтролера ESP8266

ESP8266 інтегрований 32-бітним процесор RISC Tensilica L106, який забезпечує наднизьке енергоспоживання та досягає максимальної тактової частоти 160 МГц. Операційна система реального часу (RTOS) та стек Wi–Fi дозволяють використовувати 80% обчислювальної потужності для програмування та розробки користувацьких додатків. Процесор містить наступні інтерфейси:

– програмовані інтерфейси RAM/ROM (iBus), які можуть бути підключені до контролера пам’яті та використовуватись для доступу до флеш–пам’яті;

– інтерфейс Data RAM (dBus), який може бути підключений до контролера пам’яті;

– інтерфейс AHB, який може використовуватися для доступу до регістрів.

ESP8266 інтегрований контролером пам’яті та блоками пам’яті, включаючи SRAM і ROM. Мікроконтролер може отримувати доступ до блоків пам’яті через інтерфейси iBus, dBus і AHB. Усі блоки пам’яті можуть бути доступні на запит, при цьому арбітр пам’яті вирішуватиме порядок виконання

відповідно до часу, коли ці запити отримані процесором. На рис. 2.5 зображено функціональну блокову діаграму мікроконтролера.

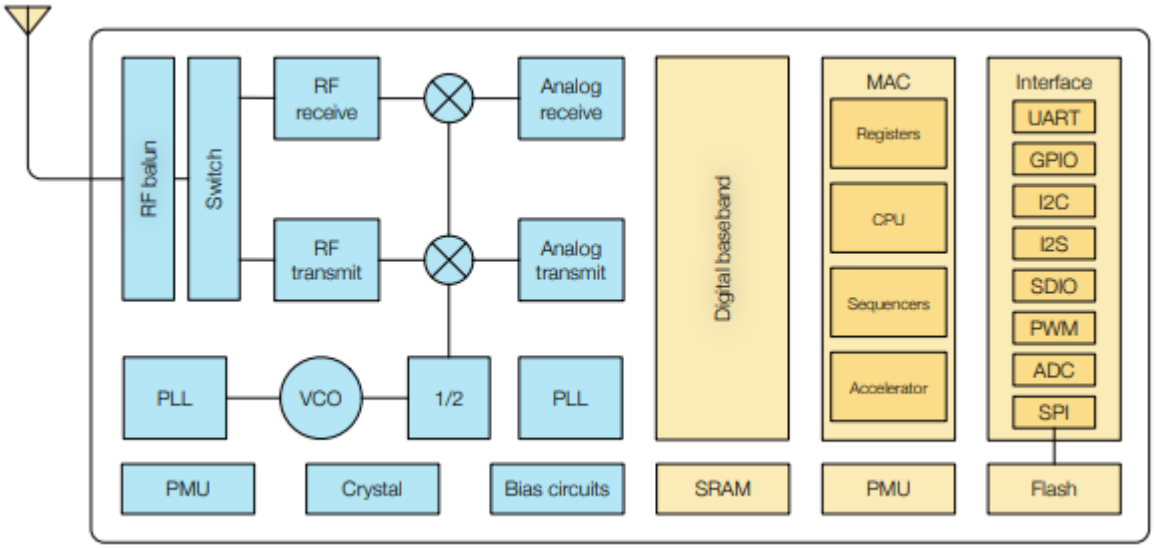

Рисунок 2.5 – Функціональна блокова діаграма мікроконтролера

Згідно з поточною версією SDK, обсяг доступної користувачам SRAM розподіляється наступним чином:

– розмір RAM < 50 кБ, тобто, коли ESP8266 працює в режимі станції (Station mode) та підключається до маршрутизатора, максимальний доступний простір для програмування у Heap + Data розділі становить близько 50 кБ;

– SoC (система на чіпі) немає програмованої ROM. Таким чином, користувацька програма повинна зберігатися у зовнішній SPI флеш-пам'яті;

ESP8266 використовує зовнішню SPI флеш-пам'ять для зберігання користувацьких програм і теоретично підтримує до 16 МБ обсягу пам'яті.

Мінімальний обсяг флеш–пам'яті для ESP8266 наведено нижче:

– вимкнене OTA (оновлення по повітрю): щонайменше 512 кБ;

– увімкнене OTA (оновлення по повітрю): щонайменше 1 МБ.

Високочастотний годинник на ESP8266 використовується для керування як передавачами, так і приймачами. Цей годинник генерується від внутрішнього кварцового генератора і зовнішнього кристала. Частота

кристала може варіюватися від 24 МГц до 52 МГц. Внутрішнє калібрування в кварцовому генераторі забезпечує можливість використання широкого діапазону кристалів, проте якість кристала залишається важливим фактором для забезпечення розумного фазового шуму та хорошої чутливості Wi–Fi. Розпіновку мікроконтролера ESP8266 зображено в табл. 2.1.

Таблиця 2.1 – Розпіновка мікроконтролера ESP8266

Назва піна	Функція
A0	ADC0 / TOUT
G	GND
VU	VUSB
S3	GPIO10 / SDD3
S2	GPIO9 / SDD2
S1	MOSI / SDD1
SC	CS / SDCMD
S0	MISO / SDD0
SK	SCLK / SDCLK
G	GND
3V	3.3V
EN	EN
RST	RST
VIN	Vin
D0	GPIO16 / USER / WAKE
D1	GPIO5
D2	GPIO4
D3	GPIO0 / FLASH
D4	GPIO2 / TXD1
D5	GPIO14 / HSCLK

Продовж. табл. 2.1

Назва піна	Функція
D6	GPIO12 / HMISO
D7	GPIO13 / RXD2 / HMOSI
D8	GPIO15 / TXD2 / HCS
RX	GPIO3 / RXD0
TX	GPIO1 TXD0

A0 (ADC0 або TOUT) – це єдиний аналоговий вхід, що підтримує напругу до 1 В. VU (VUSB) – це вхід для живлення 5 В від USB. S2 (GPIO9) – загальний цифровий вхід/вихід, який рідко використовується. S1 (GPIO10) – ще один загальний цифровий вхід/вихід, який також зазвичай не використовується. SC (SDD3, GPIO10) – використовується як SPI MOSI для підключення SD-карти. S0 (SDD2, GPIO9) – SPI MISO для підключення SD-карти. SK (SDCLK) – SPI SCK (годинник) для підключення SD-карти. G – пін заземлення. 3V3 забезпечує підключення до живлення 3.3 В. EN (Enable або CH_PD) контролює живлення чіпа. RST – пін для скидання мікроконтролера. Vin забезпечує вхідний напруговий рівень 5 В. D0 (GPIO16) – загальний цифровий вхід/вихід, який також використовується для пробудження мікроконтролера (WAKE). D1 (GPIO5) – загальний цифровий вхід/вихід. D2 (GPIO4) – загальний цифровий вхід/вихід. D3 (GPIO0) – загальний цифровий вхід/вихід, часто використовується для флешування мікроконтролера. D4 (GPIO2) – загальний цифровий вхід/вихід. D5 (GPIO14) – загальний цифровий вхід/вихід, може бути використаний як SPI CLK. D6 (GPIO12) – загальний цифровий вхід/вихід, який може використовуватися як SPI MISO. D7 (GPIO13) – загальний цифровий вхід/вихід, який може використовуватися як SPI MOSI. D8 (GPIO15) – загальний цифровий вхід/вихід, який може бути використаний як SPI CS. RX (GPIO3) – це пін для прийому серійних даних через UART. TX

(GPIO1) – пін для передачі серійних даних через UART [10]. Розпіновку відносно плати зображено на рис. 2.6.

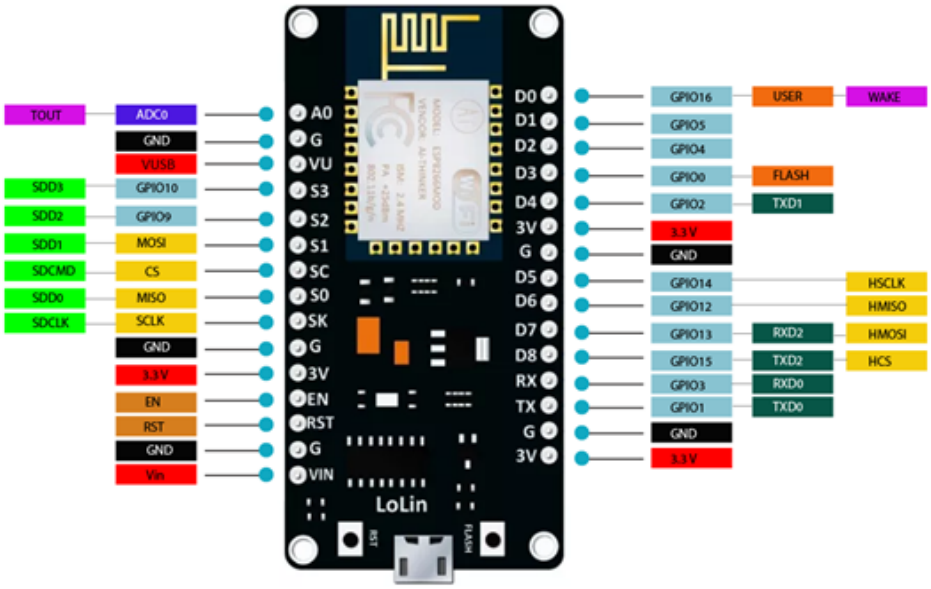

Рисунок 2.6 – Розпіновка мікроконтролера ESP8266

2.2.2 Сенсор наявності газу MQ2

MQ-2 – це популярний сенсор газу, що використовується для виявлення різних горючих газів, таких як пропан, метан, бутан, а також дим. Він підходить для застосувань у домашніх системах безпеки, промислових установках, автомобільних системах та інших проектах, де потрібен контроль наявності газу або диму в повітрі. Зовнішній вигляд MQ2 зображено на рис. 2.7.

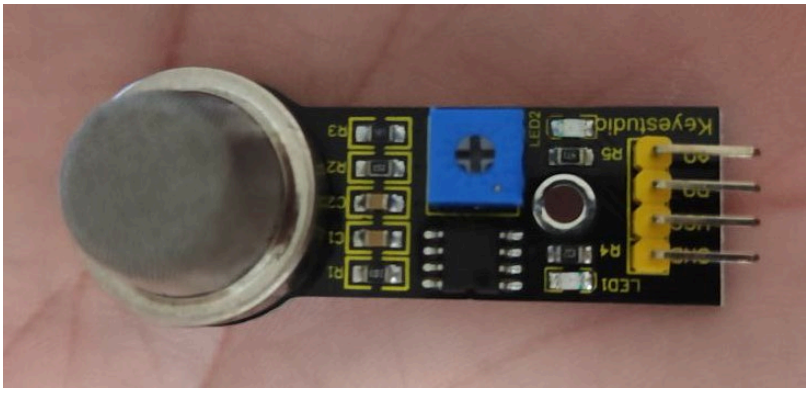

Рисунок 2.7 – Зовнішній вигляд MQ2

Основні характеристики сенсора:

1) Тип сенсора – резистивний сенсор (серія MQ);
2) Вимірювані гази: Водень, пропан, метан, бутан, окис вуглецю та дим;
3) Діапазон вимірювання:

– Метан: 200 – 10000 ppm;
– Пропан/бутан: 300 – 10000 ppm;
– Водень: 100 – 2000 ppm;
– Окис вуглецю: 20 – 2000 ppm;
– Дим: 300 – 10000 ppm.

4) Основним елементом сенсора є оксид металу, що змінює свою провідність у присутності газів;
5) Робоча напруга 5 В (VCC);
6) Споживання струму близько 150 мА;
7) Робоча температура: -20°C до 50°C;
8) Аналоговий та цифровий вихід (напруга пропорційна концентрації газу) [12].

MQ–2 використовує хімічно активний шар на основі діоксиду олова, який в нормальних умовах має низьку провідність. Коли присутні певні гази, хімічні реакції на поверхні змінюють провідність цього шару. Ця зміна провідності реєструється і перетворюється в електричний сигнал, який можна виміряти [11].

2.2.3 Сенсор вологості та температури DHT11

Сенсор DHT11 – це широко розповсюджений сенсор, що вимірює вологість та температуру. Він простий у використанні та відносно дешевий, що робить його популярним вибором для багатьох проєктів, зокрема у домашній автоматизації. Зовнішній вигляд DHT11 зображено на рис. 2.8.

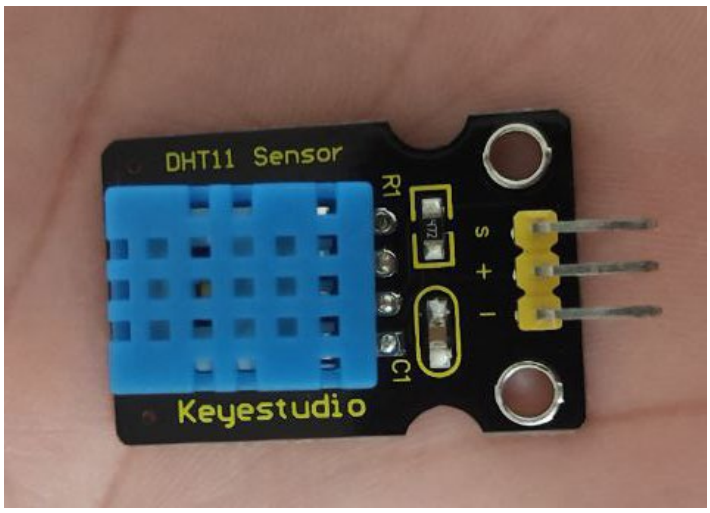

Рисунок 2.8 – Зовнішній вигляд DHT11

Основні характеристики сенсора вологості та температури DHT11:

1) Діапазон вимірювання температури від 0 до +50 °C (точність ±2 °C);
2) Діапазон вимірювання вологості від 20% до 90% відносної вологості (точність ±5% RH);
3) Живлення: 3.3 – 5.5 В;
4) Час опитування та оновлення даних приблизно кожні 1-2 секунди;
5) DHT11 складається з двох основних компонентів:

– NTC термістор для вимірювання температури;
– ємнісний сенсор вологості для вимірювання вологості.

6) DHT11 має 4 контакти:

– VCC – живлення;
– GND – земля;
– Data – передача даних;
– Not Connected – не використовується (іноді взагалі відсутній).

Для отримання даних з сенсора, використовується однодротовий протокол обміну даними. Це означає, що обмін даними здійснюється через один сигнальний контакт [14].

2.2.4 Резистор

Резистор — це пасивний електронний компонент, який обмежує або регулює потік електричного струму в колі. Зовнішній вигляд резистора зображено на рис. 2.9.

Рисунок 2.9 – Зовнішній вигляд резистора

Він є одним із найпоширеніших елементів у електронних схемах і виконує кілька важливих функцій. Основні функції резисторів:

– резистори використовуються для контролю кількості струму, що проходить через інші компоненти, запобігаючи їх пошкодженню від надлишкового струму;

– вони допомагають у поділі напруги між різними частинами схеми, забезпечуючи стабільне функціонування компонентів;

– резистори можуть перетворювати електричну енергію в теплову, що використовується, наприклад, у нагрівальних елементах.

Резистори зазвичай маркуються кольоровими смугами або цифрами, які вказують на їх номінальний опір і допустиме відхилення. Кожен колір відповідає певному числу, і значення читаються послідовно, щоб визначити опір.

Резистори можуть бути з'єднані в схемі послідовно або паралельно. У послідовному з'єднанні їхні опори додаються, а в паралельному – загальний опір зменшується.

Резистори є критично важливими для належного функціонування електронних пристроїв, забезпечуючи контроль струму та напруги у схемах, а також допомагаючи в енергетичному управлінні та захисті компонентів.

2.2.5 Світлодіод

Світлодіод (LED – Light Emitting Diode) – це напівпровідниковий пристрій, який випромінює світло, коли через нього проходить електричний струм. Світлодіоди є основою сучасних освітлювальних технологій і широко застосовуються в багатьох галузях, від індикаторів до складних систем освітлення.

Основні компоненти світлодіода:

– напівпровідниковий кристал – це основний матеріал світлодіода який створює світло;

– металеві електроди, які підключають світлодіод до електричної схеми;

– корпус захищає напівпровідниковий кристал і допомагає в розсіюванні тепла. Також корпус може фокусувати або розсіювати світло.

Зовнішній вигляд світлодіодів зображено на рис. 2.10.

Рисунок 2.10 – Зовнішній вигляд світлодіодів

2.3 Розробка електронно–принципової схеми підключень

Спочатку створимо проект в програмі Altium Designer, після цього в проекті створимо бібліотеку елементів (рис. 2.11). Після цього зайдемо в створену бібліотеку і будемо створювати елементи. Створимо схеми для мікроконтролера ESP8266, сенсора газу MQ2, сенсора температури та вологості DHT11, світлодіода, резистора та конденсатора.

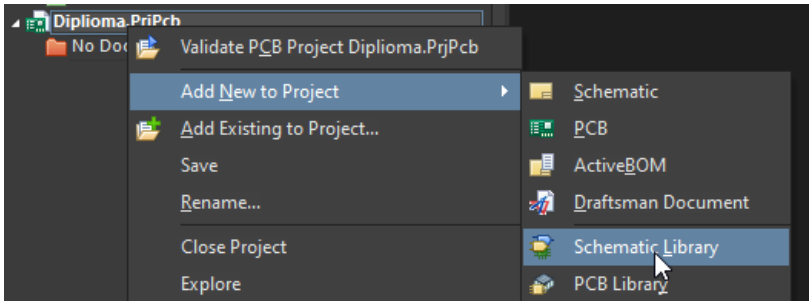

Рисунок 2.11 – Створення бібліотеки елементів

Щоб створити схему в бібліотеці потрібно спочатку нажати кнопку «Add». Переназвемо створений файл на один з потрібних нам компонентів. В відкритому пустому полотні за допомогою інстурментів створимо схему для створеного елемента (рис. 2.12).

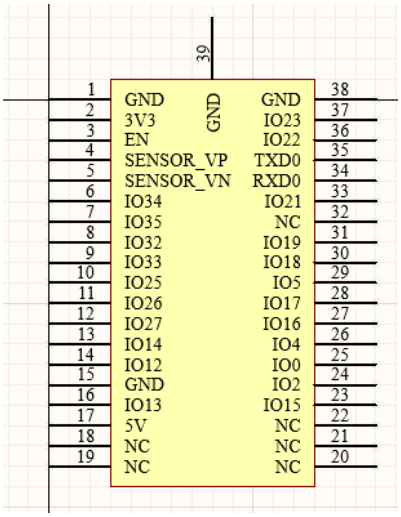

Рисунок 2.12 – Створення схеми мікроконтролера ESP8266

Таким чином створимо решту необхідних нам схем.

Створимо бланк схеми. Для цього нажимаємо по проєкту правою кнопкою миші, додати новий елемент проекту і вибираємо схему (рис. 2.13).

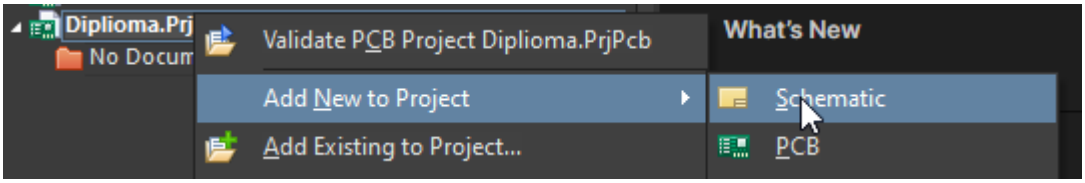

Рисунок 2.13 – Створення бланку схеми в проекті

Після цього заходимо в створену раніше бібліотеку, вибираємо потрібний нам елемент та нажимаємо кнопку “Place”. Таким методом добавимо всі потрібні нам елементи.

Після додавання на бланк схеми всіх елементів підключимо всі елементи згідно табл. 2.1. В результаті ми отримаємо електронно-принципову схему по якій ми зможемо зібрати систему вживу. Електрично-принципову схему можна розглянути на рис. 2.14.

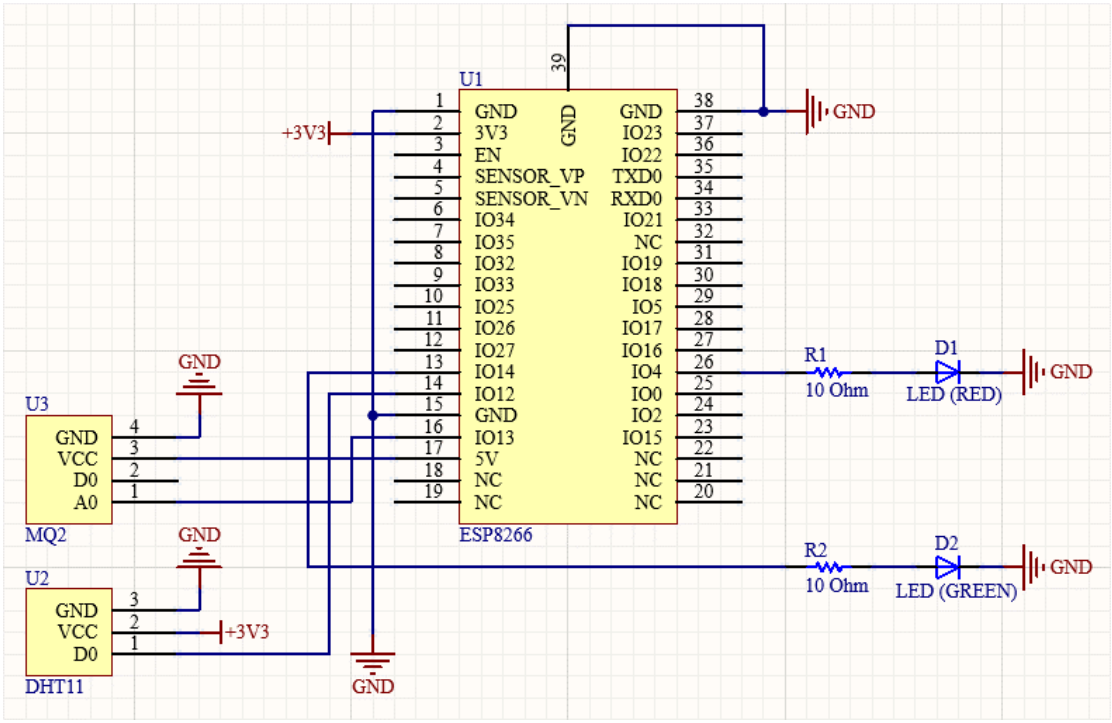

Рисунок 2.14 – Розроблена електронно-принципова схема

2.4 Конструювання системи

Згідно електрично-принципової схеми тепер складемо систему, звіряючись з табл. 2.1. Підключаємо живлення та заземлення в спеціальні піни на breadboard, підключаємо до breadboard сенсори MQ2, DHT11, резистори та світлодіоди. З’єднуємо їх до необхідних пінів: DHT11 до D6, MQ2 до D7, зелений світлодіод до D5, червоний світлодіод до D2.

З`єднувати піни можна за допомогою jump-wire. Є різні види jump-wire. Вони поділяються за методом підключення: male-male, female-male, female-female. Зовнішній вигляд jump-wire зображено на рис. 2.15. Тепер складемо схему. Складена схема зображена на рис. 2.16.

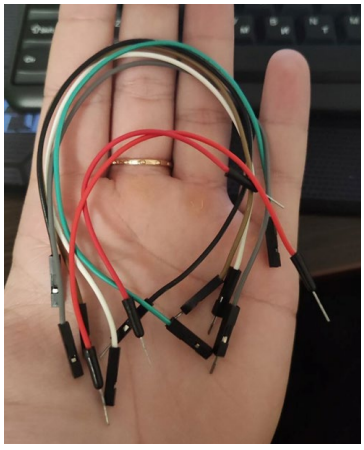

Рисунок 2.15 – Зовнішній вигляд jump-wire

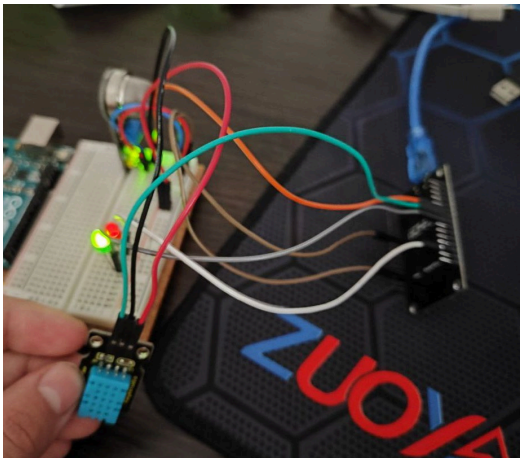

Рисунок 2.16 – Складена система

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ ЗБОРУ ТА ЛОГУВАННЯ ПОКАЗНИКІВ СЕНСОРІВ ДЛЯ ПОЖЕЖНОЇ СИГНАЛІЗАЦІЇ

3.1 Блок-схема алгоритму

Перед написанням коду потрібно скласти блок-схему алгоритму. Блок-схема алгоритму зображена на рис. 3.1.

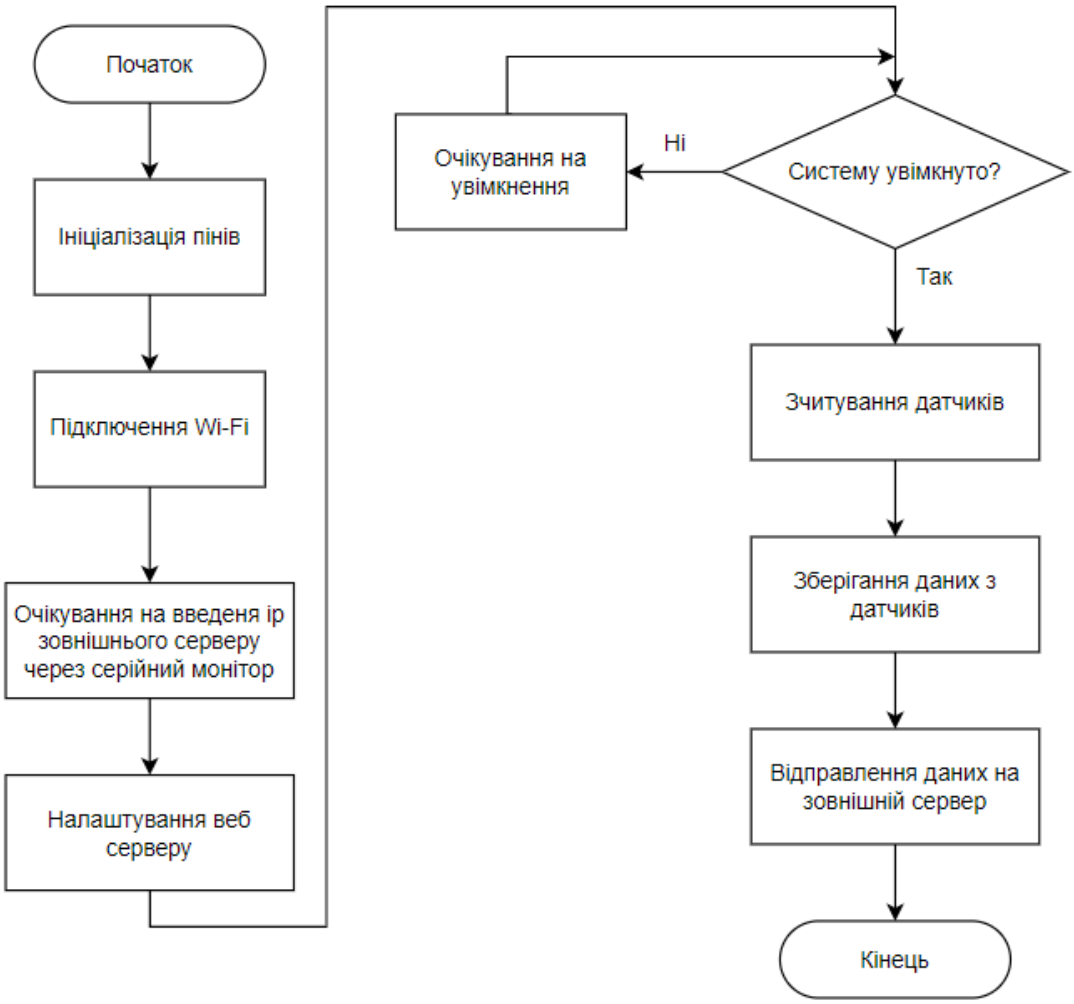

Рисунок 3.1 – Блок-схема алгоритму

Початок: Це початкова точка алгоритму.

Ініціалізація пінів: Процес починається з ініціалізації пінів підключень.

Підключення Wi-Fi: Наступним кроком є підключення системи до мережі Wi-Fi, що необхідно для здійснення передачі даних.

Очікування на введення IP зовнішнього сервера через серійний монітор: Система чекає введення IP зовнішнього сервера через серійний монітор.

Налаштування веб серверу: Алгоритм продовжується налаштуванням веб-сервера, необхідного для хостингу та передачі даних по мережі.

Система увімкнена?: Це точка прийняття рішення. Якщо система активована, вона переходить до наступного кроку; в іншому випадку повертається до очікування активації.

Очікування на увімкнення: На цьому етапі система чекає сигналу активації. Якщо на веб-сервері нажати на кнопку активації то на систему надішлеться сигнал для зчитування даних.

Зчитування сенсорів: Після активації система зчитує дані з сенсорів.

Збереження даних з сенсорів: Зчитані дані зберігаються для подальшого надсилання.

Відправлення даних на зовнішній сервер: Збережені дані потім надсилаються на зовнішній сервер для відображення в реальному часі та зберіганні в базі даних.

Кінець: Це позначає закінчення процесу алгоритму.

Ця блок-схема представляє типовий процес системи збору даних, де піни ініціалізуються, система підключається до мережі, чекає введення даних від сенсорів, зчитування даних з сенсорів активується при отриманні сигналу про включення системи, збирає та зберігає дані з сенсорів, а потім передає дані на зовнішній сервер.

3.2 Обґрунтування вибору середовища розробки

Для розробки комп'ютеризованої системи збору та логування показників сенсорів для пожежної сигналізації було обрано використання декількох середовищ розробки, кожне з яких виконує свою специфічну роль у процесі створення системи.

Arduino IDE було обрано як основне середовище для програмування ESP8266. Цей вибір пояснюється наступними факторами:

– Arduino IDE надає інтуїтивно зрозумілий інтерфейс, що значно полегшує процес написання, компіляції та завантаження коду в мікроконтролер. Це особливо корисно на етапі швидкої розробки прототипів;

– Arduino IDE підтримує велику кількість мікроконтролерів, включаючи ESP8266, який є основою системи збору даних у цьому проекті. Доступність різноманітних бібліотек та прикладів коду дозволяє швидко інтегрувати різні сенсори та інші компоненти;

– величезна спільнота користувачів Arduino забезпечує доступ до великої кількості ресурсів, обговорень та прикладів коду, що може бути корисним при виникненні труднощів або потребі в додаткових функціональностях.

Для порівняння, альтернативним середовищем могло б бути PlatformIO. Воно також підтримує ESP8266 та пропонує більш просунуті можливості, такі як багатоплатформна розробка, інтеграція з різними текстовими редакторами та покращене управління бібліотеками. Проте, для цього проекту було вирішено зупинитися на Arduino IDE через його простоту та доступність.

Для розробки веб-сторінки та серверу бази даних було обрано WebStorm. Це середовище є оптимальним для виконання таких завдань завдяки наступним характеристикам:

– WebStorm забезпечує комплексний набір інструментів для написання та налагодження JavaScript-коду, що включає підтримку таких

технологій, як Node.js, Express.js та інших бібліотек, що можуть бути використані для створення серверу та інтерфейсу користувача;

– середовище розробки надає інтелектуальні підказки та автозавершення коду, що допомагає зменшити кількість помилок і прискорити процес написання програм;

– WebStorm має вбудовану підтримку Git та інших систем контролю версій, що значно спрощує процес керування версіями коду та співпраці з іншими розробниками.

Як альтернатива, для розробки веб-сторінок та серверної частини могла бути використана Visual Studio Code (VS Code). Це середовище розробки є дуже популярним завдяки своїй легкості, широкій підтримці розширень і великій спільноті користувачів. Проте, WebStorm було обрано через його більш інтегровану підтримку для JavaScript та зручні інструменти для професійної веб-розробки.

3.1.1 Arduino IDE

Arduino IDE (Integrated Development Environment) — це програмне середовище, яке використовується для розробки та програмування мікроконтролерів на платформах Arduino. Воно надає простий і зрозумілий інтерфейс, де користувач може писати, компілювати та завантажувати програмний код безпосередньо на мікроконтролер. Основна частина інтерфейсу – редактор коду, де пишуться програми на мові C++. Редактор підтримує підсвічування синтаксису та базове автозаповнення, що полегшує процес написання коду.

Вгорі вікна IDE розташована панель інструментів з кнопками для виконання основних дій: верифікації (перевірки) коду на наявність помилок, завантаження готового коду на плату, створення нових проектів, відкриття існуючих та збереження змін. Є також кнопка для відкриття серійного

монітора, що дозволяє спілкуватися з платою через послідовний порт. Зовнішній вигляд програми зображено на рис. 3.2.

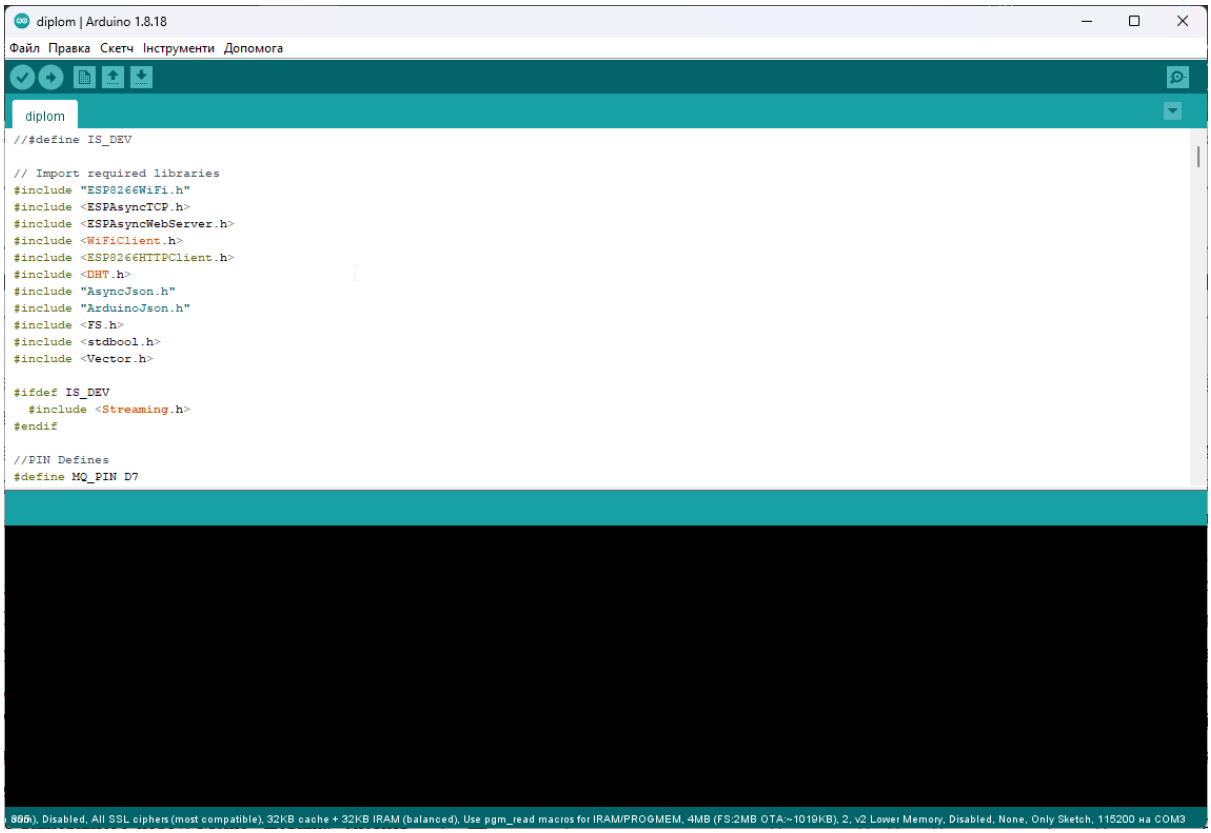

Рисунок 3.2 – Зовнішній вигляд програмного забезпечення Arduino IDE

Меню налаштувань у верхній частині включає різні опції для керування проектом та налаштуванням IDE. В меню «Файл» можна створити новий скетч, відкрити або зберегти існуючий, а також імпортувати бібліотеки. У меню «Редагування» містяться стандартні функції для редагування тексту, такі як копіювання та вставка. В меню «Інструменти» можна налаштувати параметри платформи, вибрати тип плати, з якою ви працюєте, та підключити її до відповідного порту. Меню «Допомога» містить посилання на документацію та онлайн-ресурси для отримання додаткової інформації.

Нижня частина вікна IDE відведена під консоль повідомлень, яка показує результати компіляції та завантаження, а також повідомлення про помилки та попередження. Для зручності, користувачі можуть переглядати

список файлів, які входять до поточного проекту, в окремому вікні або у лівій частині інтерфейсу.

Arduino IDE підтримує численні бібліотеки, що додають функціонал для роботи з різноманітними сенсорами, дисплеями, модулями бездротового зв'язку, тощо. Завдяки менеджеру бібліотек, користувачі можуть легко встановлювати та керувати бібліотеками. Платформа також підтримує розширення та плагіни, які додають нові можливості для IDE.

Arduino IDE є крос-платформним інструментом, доступним для Windows, macOS та Linux, що робить його зручним для широкого кола користувачів. Робота з Arduino IDE починається з написання коду (скетча), який визначає поведінку плати. Програма зазвичай складається з двох основних функцій: setup(), яка виконується один раз при запуску, і loop(), яка виконується безперервно. Після написання коду його можна компілювати, натиснувши кнопку «Перевірити», що перевіряє програму на наявність помилок. Після успішної компіляції код можна завантажити на плату через USB, натиснувши кнопку «Вивантажити». Для налагодження та комунікації з платою використовується серійний монітор, що дозволяє переглядати та відправляти дані через послідовний порт.

Arduino IDE ідеально підходить для початківців завдяки своїй простоті та інтуїтивному інтерфейсу. Її легко освоїти навіть тим, хто раніше не працював з програмуванням мікроконтролерів. Широка спільнота користувачів, велика кількість ресурсів, документації та прикладів роблять її потужним інструментом для хобі-розробників та освітніх програм. Проте, IDE може здатися обмеженою для більш просунутих проектів, які потребують кращих можливостей управління проектами та налагодження. У таких випадках може знадобитися використання більш функціональних середовищ розробки, таких як Visual Studio Code або PlatformIO. Незважаючи на це, Arduino IDE залишається популярним вибором для швидкого створення прототипів та навчання основам програмування мікроконтролерів [16].

3.1.2 WebStorm

WebStorm – це потужне інтегроване середовище розробки, створене компанією JetBrains, спеціально для веб-розробників. Це програмне забезпечення підтримує сучасні технології, мови програмування та фреймворки, що робить його універсальним інструментом для створення веб-додатків. Однією з основних особливостей WebStorm є підтримка широкого спектра мов програмування, включаючи JavaScript, TypeScript, HTML, CSS та інших. Крім того, він підтримує серверний JavaScript через Node.js і може працювати з іншими популярними мовами, такими як Dart, PHP та Python, завдяки додатковим плагінам.

WebStorm надає інтелектуальний редактор коду з розширеним автодоповненням, інтеграцією з документацією та навігацією по коду. Він пропонує підсвічування синтаксису, перевірку помилок у реальному часі та пропозиції щодо виправлення коду. Це середовище розробки також підтримує популярні фронтенд-фреймворки, такі як Angular, React і Vue.js, а також CSS-препроцесори (Sass, Less), системи збірки (Webpack, Gulp, Grunt) і менеджери пакетів (npm, Yarn).

WebStorm має вбудовані інструменти для налагодження коду, включаючи підтримку налагодження Node.js та клієнтського JavaScript, а також можливості для тестування коду з використанням популярних фреймворків, таких як Jest, Mocha та Karma. WebStorm інтегрується з системами контролю версій, включаючи Git, SVN та Mercurial, пропонуючи інтуїтивний інтерфейс для управління репозиторіями, злиттям та вирішенням конфліктів.

Ще однією важливою функцією WebStorm є потужні інструменти для рефакторингу коду, такі як перейменування, переміщення файлів, виділення функцій та інші. Вони також включають можливості для автоматичного впорядкування імпортів та оптимізації коду. Програмне забезпечення підтримує шаблони та фреймворки, такі як Pug (Jade), EJS та Handlebars, і

дозволяє переглядати та дебагувати шаблони у реальному часі завдяки інтеграції з веб-серверами. Зовнішній вигляд програми зображено на рис. 3.3.

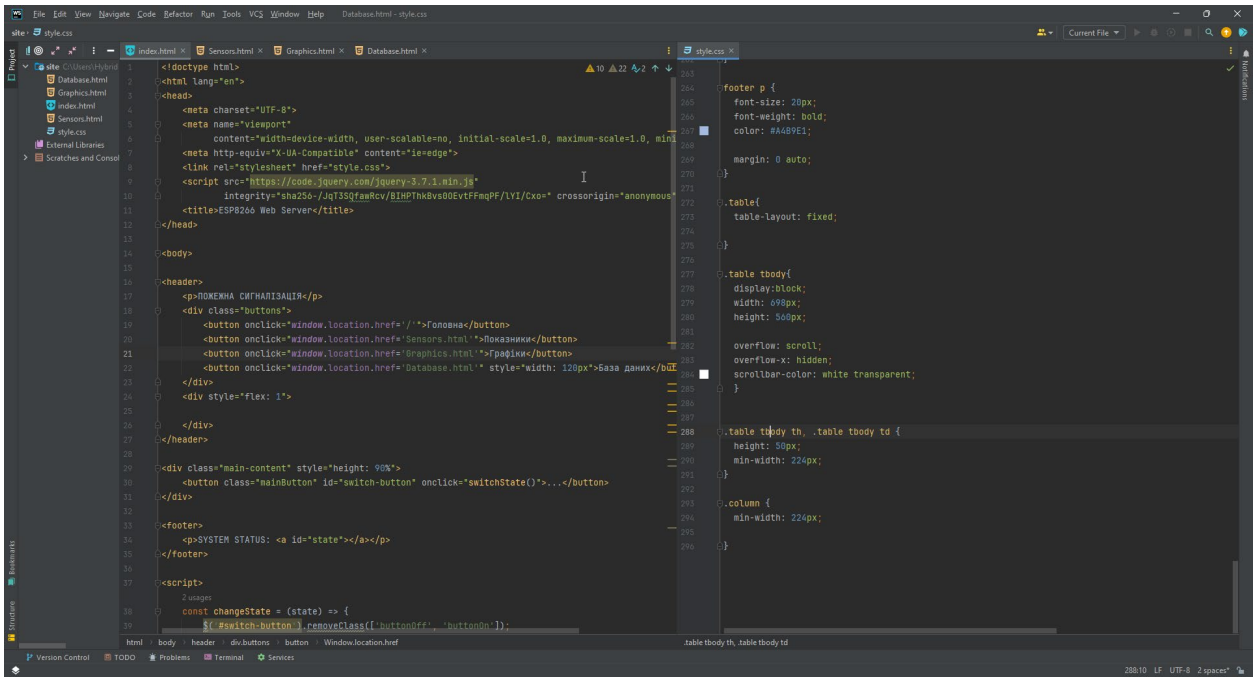

Рисунок 3.3 – Зовнішній вигляд програмного забезпечення WebStorm

WebStorm пропонує зручний та настроюваний інтерфейс, який дозволяє адаптувати середовище під свої потреби, включаючи підтримку світлих і темних тем та можливість налаштування гарячих клавіш. Велику увагу приділено підтримці плагінів, що дозволяє розширювати функціональність IDE або інтегрувати його з іншими інструментами.

Основними перевагами WebStorm є його інтелектуальні функції, які автоматизують завершення коду, коригування помилок у реальному часі та забезпечують надійний рефакторинг, а також його ефективність, що прискорює процес розробки завдяки інтеграції з системами контролю версій, збірниками завдань та тестовими фреймворками. WebStorm можна налаштувати для будь-яких потреб розробника, що робить його ідеальним вибором для створення сучасних веб-додатків [17].

3.2 Реалізація системи в програмному забезпеченні

Спочатку будемо реалізовувати веб-сторінку в програмному забезпеченні WebStorm. Створимо проект та файл «index.html». Відкриємо створений файл та почнемо писати код. WebStorm вже створив за нас бланк сторінки.

На даний момент це просто пуста біла сторінка. Напишемо стилі для основних тегів нашої сторінки, тобто «html» та «body» (рис. 3.4)..

```
html, body {
  height: 100%;
  width: 100%;
  margin: 0;
  padding: 0;
}

body {
  background-image: url("https://i.imgur.com/erPWads.jpeg");

  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
}
```

Рисунок 3.4 – Лістинг CSS-стилів для основних тегів веб-сторінки

Для тегів «html» та «body» використаємо теги «height» та «width» зі значеннями «100%» щоб наша веб-сторінки займала все 100% місця як по висоті так і по ширині. Також використаємо теги для зовнішнього та внутрішнього відступу – «margin» та «padding» аналогічно. Напишемо значення для відступів по «0».

Для тегу «body» окремо допишемо ще декілька стилів. Напишемо тег «background-image» для задання заднього фону для веб-сторінки. Використаємо тег «background-repeat» зі значенням «no-repeat» щоб фотографія заднього фону не повторювалась якщо вона буде замала для якогось розширення. Застосуємо тег «background-position» зі значенням

«center» – це потрібно щоб фотографія заднього фону все центрувалась відносно сторінки. Використаємо також тег «background-size» зі значенням «cover» щоб наша фотографія заднього фону сама розширялась і заповняла 100% веб-сторінки

Напишемо код для «хедеру» нашого сайту (рис. 3.5). В «хедері» буде знаходитись надпис «Пожежна сигналізація» та навігаційні кнопки які будуть пересилати на інші веб-сторінки.

```
<header>
    <p>ПОЖЕЖНА СИГНАЛІЗАЦІЯ</p>
    <div class="buttons">
        <button onclick="window.location.href='/'">Головна</button>
        <button onclick="window.location.href='Sensors.html'">Показники</button>
        <button onclick="window.location.href='Graphics.html'">Графіки</button>
        <button onclick="window.location.href='Database.html'" style="width: 120px">База даних</button>
    </div>
    <div style="flex: 1">

    </div>
</header>
```

Рисунок 3.5 – Лістинг «хедеру» веб-сторінки

В даному лістингу є декілька основних тегів з атрибутами. Основним тегом є тег «header» – він використовується за принципом тегу «div», тобто як тег групування елементів.

Спочатку використовується тег «p». Цей тег потрібен для написання тексту або абзацу.

Дальше використовуємо тег «div» – це тег контейнер. Використовується для групування елементів. Для цього тегу вкажемо атрибут «class» з значенням «buttons». Цей атрибут може використатись не тільки для цього тегу, а і до багатьох інших. Він використовується для з'єднання тег з CSS-кодом для стилізації якогось елементу на веб-сторінці.

Також додаємо для «хедеру» кнопки навігації. Для цього використаємо тег «button». Цей тег створює кнопку для навігації по веб-сторінці, надсилання запиту на сервер або для якогось функціоналу на сайті. Додамо для цього тегу

атрибут «onClick» – він використовується для запуску функції яку ми напишемо в значеннях цього атрибуту. Використаємо функціонал для переходу між веб-сторінками, для цього напишемо значення «window.location.href=’шлях до файлу веб-сторінки’». Якщо ми вкажемо як шлях «/», то нас буде пересилати на файл з назвою «index.html».

Розглянемо CSS-код для елементів групового тегу та тегу тексту для «хедеру» (рис. 3.6).

```
header {
  display: flex;
  flex-direction: row;
  width: 100%;

  margin: 0;
}

header p {
  font-size: 30px;
  font-weight: bold;
  flex: 1;

  color: #6F8CC3;

  font-family: "Lato", sans-serif;
  align-self: center;

  margin: 0 0 0 20px;
}
```

Рисунок 3.6 – Лістинг стилів для групового тегу та тегу тексту «хедеру»

Спочатку напишемо стилі для групового тегу «header». Використаємо стиль подання блоків за допомогою тегу «display» та значенням «flex». Це потрібно для того щоб наші елементи в цьому тегу мали можливість розтягуватись та стискатись за потребою. Тег «flex-direction» з значенням «row» використовується щоб наші блоки в групі виставлялись в рядок по черзі (якщо б використовувалось значення «column» тоді всі елементи виставлялись в стовпчик). Тег «width» з значенням «100%» використовується для того, щоб ця група елементів займала 100% можливої площі веб-сторінки. Тег «margin» використовується для задання зовнішнього відступу від блоку. Ми

використовуємо значення 0, тому що нам не потрібно зовнішнього відступу для «хедеру».

Напишемо тепер стилі для тексту «хедера». Задаємо стиль «font-size» з значенням «30px» – це розмір тексту. Тег «font-weight» використовується для завдання стилю тексту (курсив, жирний, з підкресленням, простий). Використаємо наступний стиль для тексту – жирний. Задаємо тег «flex» з значенням «1» – це потрібно для того, щоб наш блок з текстом займав 1/3 «хедеру». За допомогою тегу «color» та значенням «#6F8CC3» задаємо колір для нашого тексту. За допомогою тегу «font-family» та значенням «"Lato", sans-serif» задаємо сімейство шрифт та стиль шрифту. Тегом «align-self» та значенням «center» зробимо текст по центру блоку в якому він знаходиться. Тег «margin» також можна використовувати для задання зовнішнього відступу для певної сторони. Ми використаємо відступ для лівої сторони. Відступ задається за годинниковою стрілкою, тобто наступним чином: зверху, зліва, знизу та справа. Розглянемо тепер стилі для кнопок «хедеру» (рис. 3.7).

```
.buttons button {
  background: transparent;
  border-color: transparent;
  border-bottom-width: 3px;
  border-left-width: 0;
  border-right-width: 0;
  border-top-width: 0;

  color: #6F8CC3;
  font-size: 20px;

  font-family: "Lato", sans-serif;

  transition-duration: .2s;
}

.buttons button:hover {
  color: #E6EAF1;
  border-color: #E6EAF1;

  border-bottom-width: 3px;
}
```

Рисунок 3.7 – Лістинг стилів для кнопок «хедеру»

Для кнопок ми використаємо наступні стилі: тег «background» зі значенням «transparent» – це використовується для задання заднього фону

кнопки в прозорий, також використаємо це значення для тегу «border-color» щоб в нашої кнопки не було на даний момент контуру. Використаємо тег «border-bottom-width» зі значенням «3px» щоб в кнопки був нижній контур висотою в 3 пікселя, решті контурів задаємо висоту контура в 0 пікселів. За допомогою тегу «color» зі значенням «#6F8CC3» задамо колір тексту в кнопках. Задамо також стилі для тексту кнопок наступними тегами: «font-size» зі значенням «20px» та «font-family» зі значенням «"Lato", sans-serif». Використаємо тег для задання часу анімації «transition-duration» зі значенням «0.2s» – це потрібно для того, щоб анімації при наведені на кнопку мінялися протягом часу який ми вказалив цьому тегу.

Для того щоб зробити анімації при наведенні на кнопку потрібно в стилях як класс написати «button:hover», після цього всередині ми можемо міняти попередньо написані стилі для кнопки. Поміняємо стилі кольору контуру з прозорого на «#E6EAF1» за допомогою тегу «border-color», також поміняємо колір тексту на такий же як і для контуру. Тепер при наведені на кнопку буде мінятись колір тексту та контуру на більш світліший. На рис. 3.8 зображений лістинг таблички.

```
<div class="main-content" style="flex-direction: column; height: 90%">
    <table>
        <thead>
            <tr>
                <td>Temperature</td>
                <td>Humidity</td>
                <td>Presence of gas</td>
            </tr>
        </thead>
        <tbody id="data-table">
            <tr...>
            <tr>
                <td>24</td>
                <td>40</td>
                <td>Yes</td>
            </tr>
            <tr...>
            <tr...>
            <tr...>
            <tr...>
            <tr...>
            <tr...>
        </tbody>
    </table>
</div>
```

Рисунок 3.8 – Лістинг таблички для веб-сторінки

Спочатку створимо груповий тег «div» з класом «main-content» – це основний клас для головного контенту всіх сторінок. Для сторінки з табличками вкажемо додаткові стилі та напишемо їх в тег «div» за домогою атрибуту «style». Нам необхідно написати наступні додаткові стилі: «flex-direction» зі значенням «column» та тег «height» зі значенням «90%».

Напишемо основний тег «table» – це основний груповий тег для таблички. В цьому тегу напишемо ще 2 основних тега: «thead» та «tbody». Вони використовуються для того щоб відокремити заголовки для стовпців таблиць та для рядків значення яких будуть мінятись. Рядок для таблички створюється за допомогою тегу «tr», а для того щоб створити стовпці для таблиць скористаємось тегом «td».

Розглянемо тепер CSS-стилі для таблиці (рис. 3.9).

```
table {
  width: 700px;
  height: 600px;

  color: white;

  background-color: rgba(14, 18, 29, 0.9);

  border: 1px solid white;
  border-collapse: collapse;

  box-shadow: 0 0 50px white;
}

tr, td {
  color: white;
  font-family: "Lato", sans-serif;

  text-align: center;
  border: 1px solid white;
}
```

Рисунок 3.9 – Лістинг CSS-стилів для таблички

Задаємо фіксовану висоту та ширину для таблички. Колір для тексту задаємо як білий, за допомогою атрибуту «rgba(14, 18, 29, 0.9)» задаємо колір заднього фону як чорний з прозорістю. За прозорість відповідає останнє число. Задаємо розмір та стиль контуру, та за допомогою тегу «border-collapse» зі значенням «collapse» заберемо відступи між клітинками таблиці. Використаємо також тег «box-shadow» зі значенням «0 0 50px white» – цей тег використовується коли ми хочемо зробити ефект тіні від якогось елементу. Перші два значення це зміщення відносно елементу по x та y аналогічно, третє значення це наскільки більшим буде тінь відносно елементу, четверте значення це колір тіні.

Таким методом створимо інші веб-сторінки. Після створення веб-сторінок можна приступати до написання коду для мікроконтролера ESP8266 в програмному забезпеченні ArduinoIDE.

Спочатку додаємо в менеджері додаткових плат посилання на json з нашою платою (ESP8266). Для цього нажимаємо Файл >> Налаштування >> URL Менеджерів додаткових плат (рис. 3.10) [18].

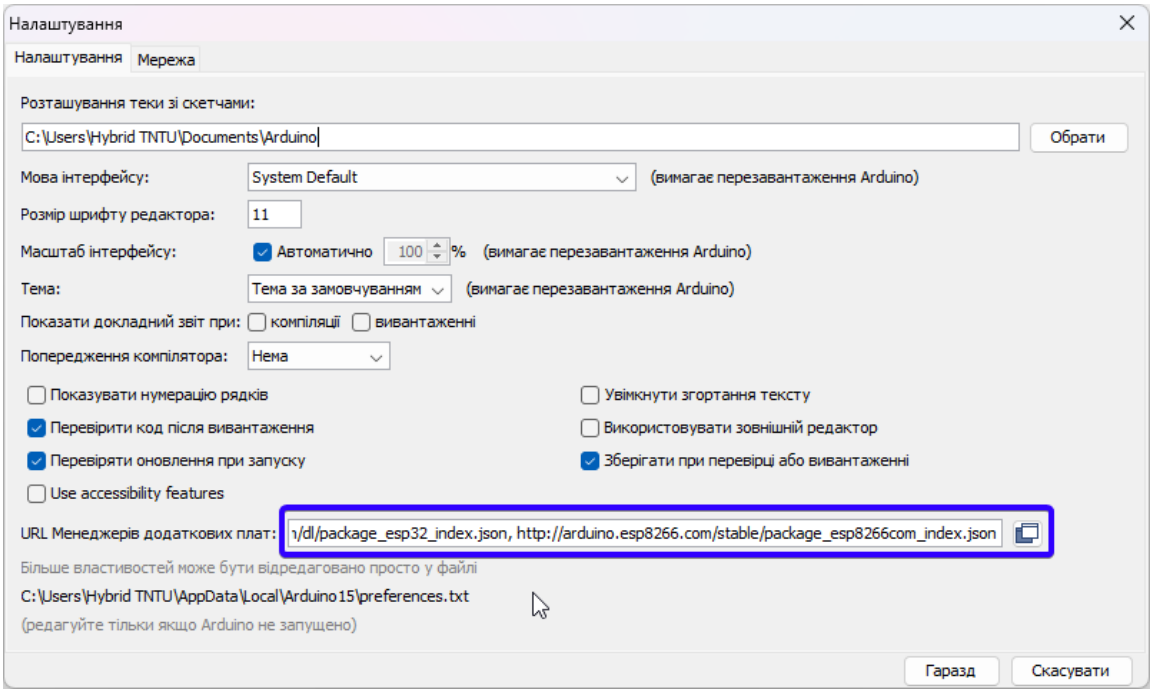

Рисунок 3.10 – Менеджер додаткових плат

Далі необхідно завантажити необхідні бібліотеки для збирання даних з сенсорів, передачі даних та Wi-Fi модуля. Для цього необхідно нажати Інструменти >> Керування бібліотеками. В пошуку вказуємо назву бібліотеки шукаємо необхідного автора бібліотеки та нажимаємо кнопку «Встановити» (рис. 3.11).

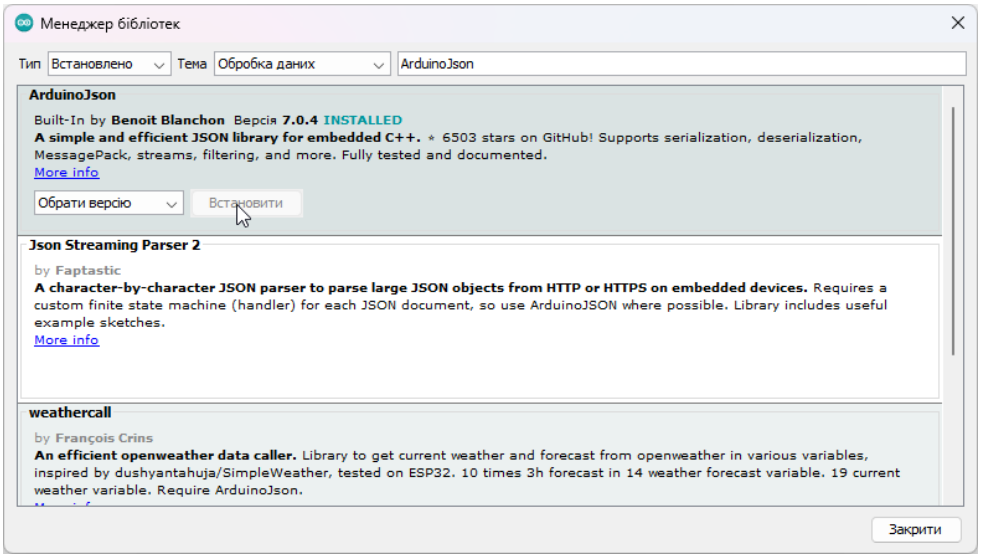

Рисунок 3.11 – Встановлення необхідних бібліотек

Після встановлення бібліотек та налаштування менеджера плат необхідно підключити ESP8266 до COM-порта. В вкладці «Інструменти» вибириаємо пункт «Плата» >> ESP8266 Boards (3.1.2) >> NodeMCU 1.0 (ESP-12E Module). Тепер ми готові писати код та прошивати нашу плату.

Спочатку підключимо всі необхідні бібліотеки (рис. 3.12).

```
#include "ESP8266WiFi.h"
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>
#include <DHT.h>
#include "AsyncJson.h"
#include "ArduinoJson.h"
#include <FS.h>
#include <stdbool.h>
#include <Vector.h>
```

Рисунок 3.12 – Лістинг підключень необхідних бібліотек

Перша бібліотека – це бібліотека яка допоможе нам підключити ESP8266 до Wi-Fi мережі [19]. Друга та третя бібліотека потрібна щоб передавати дані та розгорнути веб-сервер аналогічно [20]. Четверта та п’ята бібліотека необхідна для керування запитів Wi-Fi [22]. Шоста бібліотека необхідна для коректної роботи сенсора температури та вологості DHT11. Сьома та восьма бібліотека використовується для перетворення даних для надсилання їх на веб-сторінку [21]. Дев’ята, десята та одинадцята бібліотека необхідна для маніпулювання даними які отримуються з сенсорів [23].

Тепер згідно електрично-принципової схеми оголошуємо піни до яких ми будемо підключати сенсори та світлодіоди (рис. 3.13).

```
#define MQ_PIN A0
#define DHT_PIN D6

#define GREEN_PIN D5
#define RED_PIN D2
```

Рисунок 3.13 – Лістинг оголошення пінів для підключення

Вибираємо аналоговий пін A0 для сенсора газу MQ2, цифровий пін D6 для сенсора температури та вологості DHT11 та цифрові піни для світлодіодів D5 та D2 для зеленого та червоного світлодіодів аналогічно.

Напишемо код для підключення ESP8266 до Wi-Fi (рис. 3.14).

```
//WIFI
const char* ssid = "LiLSaiko";
const char* password = "lq2a3z4e5d";
//WIFI END

// Connect to Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi..");
}
```

Рисунок 3.14 – Лістинг коду для підключення ESP8266 до мережі Wi-Fi

Напишемо спочатку ідентифікатор та пароль Wi-Fi мережі. Запустимо ініціалізацію модуля Wi-Fi та передамо йому ідентифікатор та пароль мережі. Зробимо дальше цикл який буде писати в монітор послідовного порту «Connecting to WiFi…» поки ESP8266 не підключиться до мережі Wi-Fi. Після підключення в послідовний порт буде надіслано IP підключеного мікроконтролера. На цьому IP і буде хоститись веб-сторінка.

Напишемо код для зчитування даних з сенсора газу MQ2 (рис. 3.15).

```
bool readMQ() {
  bool gasValue = analogRead(MQ_PIN)>180;

  MQ_Vector.push_back(gasValue);
  return gasValue;
}
```

Рисунок 3.15 – Лістинг коду для зчитування даних з сенсора газу MQ2

Напишемо що значення газу буде зберігатись в змінній «gasValue». Зчитування буде здійснюватися з аналоговим зчитуванням з аналового піна. Якщо значення буде більше 180 то це буде означати що є присутність чадного диму – тобто є пожежа. Після цього це значення надсилається на веб-сторінку.

Напишемо код для зчитування даних з сенсора DHT11 (рис. 3.16).

```
int readT() {
  int temp = dht.readTemperature();

  T_Vector.push_back(temp);
  return temp;
}

int readH() {
  int hum = dht.readHumidity();

  H_Vector.push_back(hum);
  return hum;
}
```

Рисунок 3.16 – Лістинг коду для зчитування даних з сенсора DHT11

Напишемо що значення температури буде зберігатись в змінній «temp». Зчитування буде здійснюватися з цифрового піна. Після цього це значення надсилається на веб-сторінку. Зробимо це і для значень вологості, зберігатись буде в змінній «hum». Зчитування буде здійснюватися з цифрового піна. Після цього це значення також буде надсилатися на веб-сторінку.

Напишемо код для надсилання значень на веб-сторінку (рис. 3.17).

```
void sendDataToServer(int temp, int hum, bool gas) {
   WiFiClient client;
   HTTPClient http;

   char output[128];
   JsonDocument doc;
   doc["temp"] = temp;
   doc["hum"] = hum;
   doc["gas"] = gas;

   http.begin(client, "http://" + goodServerIp + "/data");
   http.addHeader("Content-Type", "application/json");

   serializeJson(doc, output);
   int httpResponseCode = http.POST(output);

   Serial.print("HTTP Response code: ");
   Serial.println(httpResponseCode);

   http.end();
}
```

Рисунок 3.17 – Лістинг коду для передачі даних на веб-сторінку

Назвемо функцію надсилання даних «sendDataToServer» та будемо передавати в цю функцію дані з сенсорів. Ця функція буде надсилати дані через HTTP-запити. Четвертий рядок це буфер для зберігання JSON-даних у вигляді рядка. П’ятий рядок потрібен для створення об'єкта JSON-документа для формування даних. Наступні три рядка додають значення в JSON-документ. Дев’ятий рядок це початок HTTP-запиту до сервера за вказаною URL-адресою. Наступний рядок додає в «хедер-відповіді» значення щоб браузер розпізнав отриманий результат як JSON структуру. Також цей код буде писати в монітор послідовного порту HTTP код відповіді.

3.3 Тестування спроектованої системи

Після написання коду можна приступити до тестування системи. Спочатку підключимо всі пристрої згідно з електрично-принциповою схемою. Прошиємо мікроконтролер та зашиємо файли в нього. Запустимо монітор послідовного порту та почекаємо поки мікроконтролер підключиться до мережі Wi-Fi. Про успішне підключення мікроконтролера буде свідчити IP-адрес веб-сторінки. Після цього підключимо мікроконтролер до бази даних яку захостимо на комп'ютері. Для підключення напишемо IP-адрес та порт в монітор послідовного порту та натиснемо кнопку «Надіслати». Про успішне підключення бази даних буде свідчити надпис «Connected» (рис. 3.18).

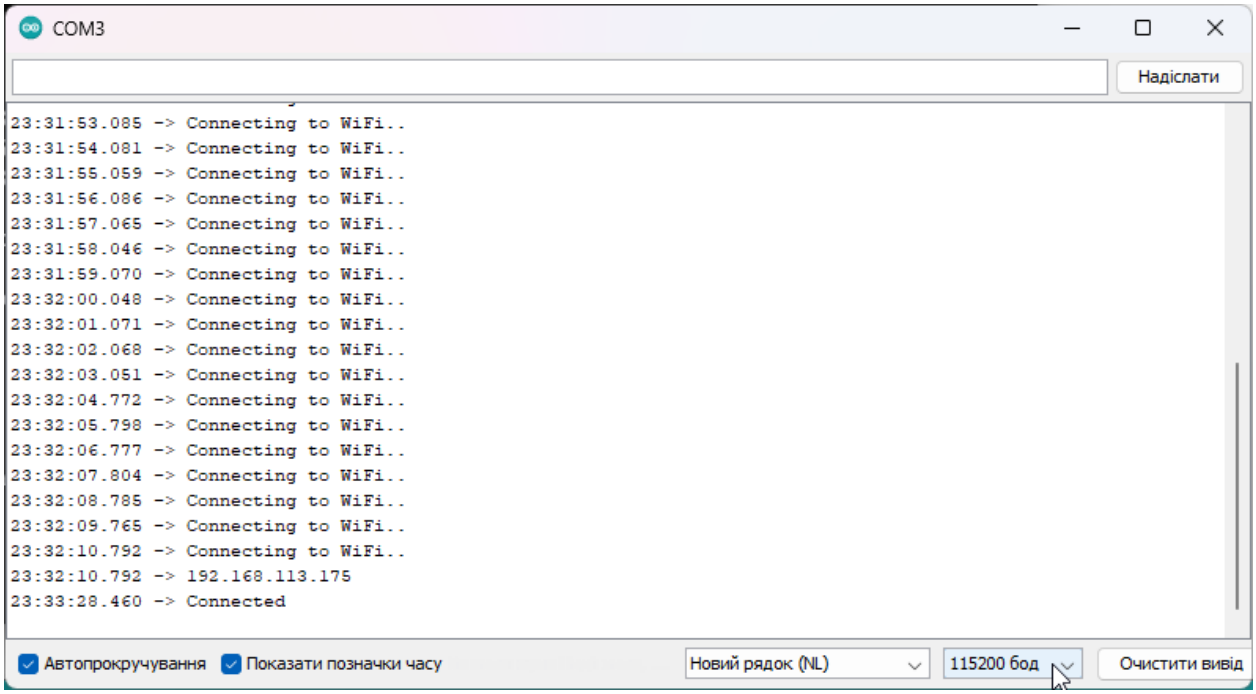

Рисунок 3.18 – Підключення мікроконтролера до мережі Wi-Fi та бази даних

Щоб запустити веб-сторінку необхідно просто написати IP підключення який отримали в моніторі послідовного порту. Після цього відкриється головна веб-сторінка з кнопкою включення або виключення системи. На кожній сторінці знизу по центру є надпис з статусом системи (рис. 3.19).

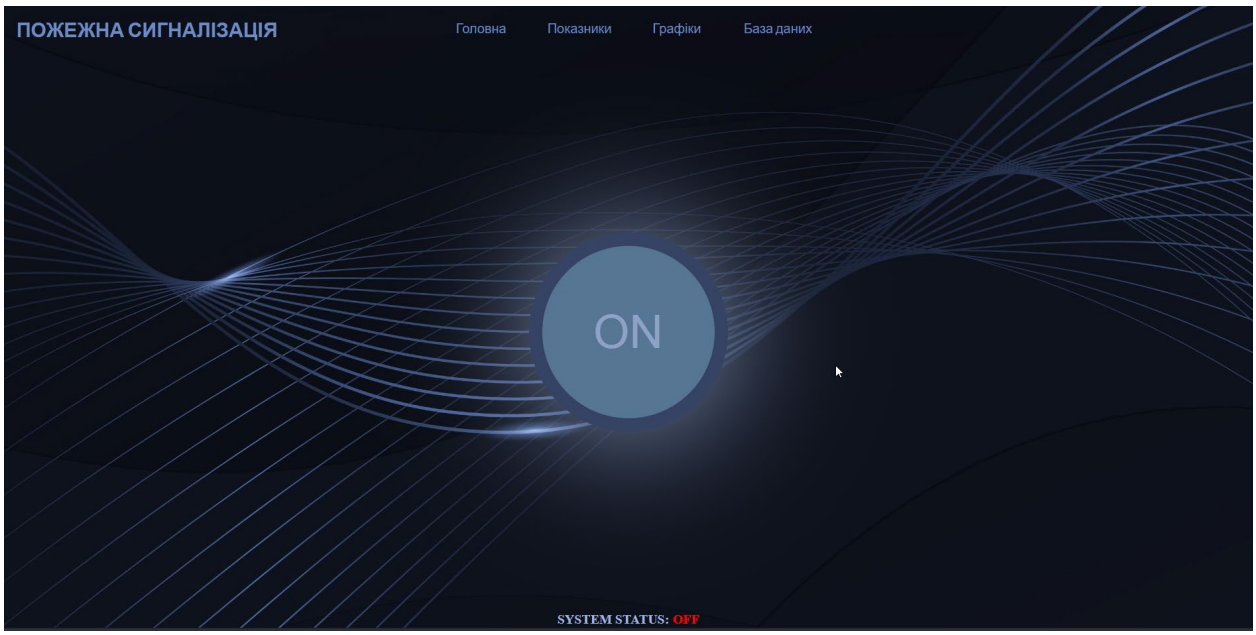

Рисунок 3.19 – Головна сторінка веб-серверу

При нажатті кнопки вона поміняє свої стилі, знизу поміняється текст та колір цього тексту що буде свідчити про включення системи (рис. 3.20).

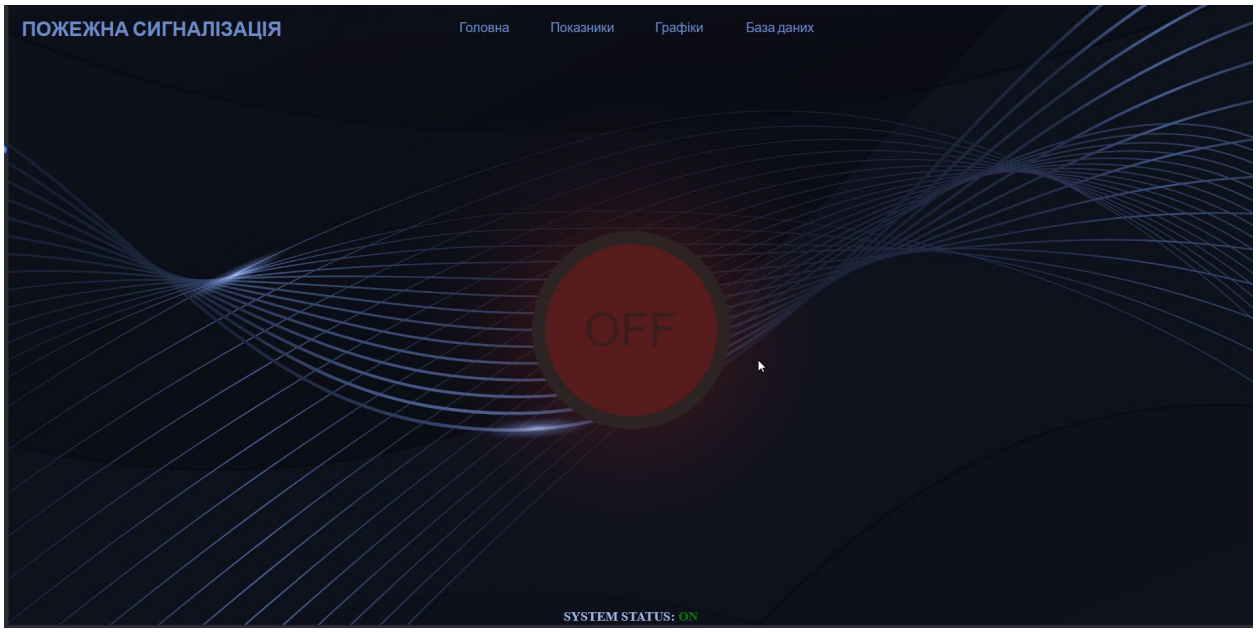

Рисунок 3.20 – Головна сторінка веб-серверу при нажатті кнопки

Тепер перейдемо на наступну сторінку за допомогою кнопки на навігаційному меню. Відкриється сторінка з даними в табличці (рис. 3.21).

Рисунок 3.21 – Сторінка з даними в табличці

Перейдем на наступну сторінку. Там будуть відображені графіки відносно значень які отримали сенсори (рис. 3.22).

Рисунок 3.22 – Сторінка з графіками відносно даних з сенсорів

Остання сторінка – це сторінка також з табличкою, але вона спочатку пуста (рис. 3.23).

Рисунок 3.23 – Сторінка з пустою табличкою

Коли ми виберемо дату та час та почекаємо, в табличку з бази даних надійдуть дані та заповнять табличку (рис. 3.24).

Рисунок 3.24 – Сторінка з табличкою та можливістю відображати дані з бази даних

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Надзвичайні ситуації, викликані пожежами, вибухами, техногенними та природніми причинами

Надзвичайні ситуації, спричинені пожежами, вибухами, техногенними та природними причинами, представляють серйозну загрозу для людського життя, майна і навколишнього середовища. Кожен з цих видів надзвичайних ситуацій має свої особливості, причини виникнення та наслідки, що вимагають специфічних підходів до їх запобігання і ліквідації.

Пожежі є одними з найпоширеніших надзвичайних ситуацій. Вони можуть виникати як у природних умовах (лісові та степові пожежі), так і в техногенному середовищі (пожежі на підприємствах, у житлових будинках, транспортних засобах). Причинами пожеж часто є людська недбалість, порушення правил пожежної безпеки, технічні несправності або природні явища, такі як блискавка. Пожежі призводять до значних матеріальних втрат, загибелі та травмування людей, забруднення навколишнього середовища димом і токсичними речовинами. Вони можуть спричинити вторинні надзвичайні ситуації, такі як вибухи чи обвали будівель [24].

Вибухи є ще одним видом надзвичайних ситуацій, які можуть мати катастрофічні наслідки. Вони можуть бути спричинені техногенними факторами, такими як несправність обладнання, порушення технологічних процесів або недотримання правил безпеки. Природні вибухи можуть статися через вулканічну діяльність або газові викиди. Вибухи можуть спричиняти великі руйнування будівель та інфраструктури, загибель і поранення людей, пожежі та екологічні катастрофи через викид токсичних речовин [24].

Техногенні надзвичайні ситуації виникають через несправності технічних систем, аварії на виробництвах, транспортні катастрофи, витоки хімічних речовин, радіаційні аварії та інші подібні інциденти. Вони можуть бути спричинені людським фактором, технічними несправностями або природними явищами. Такі ситуації характеризуються великими матеріальними втратами, значною шкодою для здоров'я та життя людей, а також негативним впливом на навколишнє середовище, витоки хімічних речовин можуть призвести до отруєння великої кількості людей і забруднення ґрунту та води [24].

Природні надзвичайні ситуації включають землетруси, повені, урагани, цунамі, зсуви, лавини та інші природні явища, що мають руйнівний характер. Вони можуть призводити до значних людських втрат, руйнування інфраструктури, порушення роботи систем життєзабезпечення, евакуації населення та тривалих економічних збитків. Природні катастрофи часто є непередбачуваними та важко контрольованими, що ускладнює заходи з їх запобігання та ліквідації наслідків, наприклад землетруси можуть зруйнувати будівлі, дороги, мости, спричинити цунамі, а також вторинні надзвичайні ситуації, такі як пожежі та витоки небезпечних речовин [24].

Для запобігання та ліквідації наслідків надзвичайних ситуацій, спричинених пожежами, вибухами, техногенними та природними факторами, необхідно впроваджувати комплексні заходи з управління ризиками. Важливою складовою є створення ефективних систем раннього виявлення та оповіщення про надзвичайні ситуації, розробка та впровадження планів евакуації, навчання населення правилам безпеки та діям у разі виникнення небезпеки. Необхідно проводити регулярний моніторинг технічного стану обладнання та інфраструктури, дотримуватися стандартів безпеки на виробництвах та в будівлях [24].

4.2 Номенклатура та необхідна кількість засобів гасіння пожежі на підприємстві.

Забезпечення належного рівня пожежної безпеки на підприємстві є надзвичайно важливим аспектом його функціонування. Це не лише обов'язкова вимога законодавства, але й запорука збереження майна, здоров'я та життя працівників. Одним із ключових елементів системи пожежної безпеки є наявність і правильне використання засобів гасіння пожежі. Основними видами таких засобів є вогнегасники, пожежні гідранти, пожежні крани, системи автоматичного пожежогасіння та пожежні водоймища. Кожен із цих засобів має своє призначення і використовується залежно від специфіки підприємства, характеру виробництва та потенційних ризиків виникнення пожежі. Вогнегасники можуть містити різні види вогнегасних речовин: водяні, пінні, порошкові, вуглекислотні та газові. Для гасіння пожеж класу А (тверді матеріали, що горять із тлінням) використовують водяні та пінні вогнегасники, а для пожеж класу В (горючі рідини) – пінні та порошкові. Вуглекислотні та газові вогнегасники використовуються для гасіння пожеж класу С (газоподібні речовини) та у випадках, коли необхідно уникнути пошкодження електронного обладнання [25].

Пожежні гідранти та крани – це засоби гасіння, які забезпечують подачу води з водопровідної мережі для боротьби з вогнем. Вони встановлюються на території підприємства у відповідності до плану пожежної безпеки, що враховує розміщення будівель і можливі джерела займання. Використання цих засобів вимагає наявності відповідного навчання у працівників, адже неправильне поводження з гідрантами може призвести до неефективного гасіння пожежі [25].

Системи автоматичного пожежогасіння є більш складними та дорогими засобами, але вони забезпечують високу ефективність гасіння пожеж на ранніх стадіях їх виникнення. До таких систем відносяться спринклерні установки,

газові системи гасіння та порошкові установки. Вибір системи залежить від характеристик об'єкта: його площі, висоти стель, виду матеріалів, що зберігаються або обробляються, та інших факторів. Системи автоматичного пожежогасіння можуть бути інтегровані з системами пожежної сигналізації, що дозволяє оперативно виявляти та ліквідовувати займання [25].

Необхідна кількість засобів гасіння пожежі на підприємстві визначається на основі кількох основних критеріїв, а саме: площа та конфігурація будівель, види виробничої діяльності, кількість і тип матеріалів та потенційні ризики виникнення пожежі.

Згідно з наказом Міністерства внутрішніх справ України від 15 січня 2018 року, у приміщеннях вогнегасники повинні бути розташовані на підлозі або закріплені в спеціальних конструкціях на стіні, на висоті не більше ніж півтора метра від поверхні підлоги. Місця їх розміщення в будівлі повинні добре проглядатися і не заважати вільній евакуації людей при пожежі. Якщо з якої-небудь причини в приміщення вогнегасники не помітні, то безпосередньо над ними на висоті в два метри розташовуються спеціальні покажчики.

Згідно з наказом № 25 міністерства внутрішніх справ України 15.01.2018 розрахунок кількості вогнегасників для будинків адміністративного та побутового призначення і громадських будинків на кожному поверсі повинні мати не менше двох переносних (порошкових, водопінних або водяних) вогнегасників з масою заряду вогнегасної речовини 5 кг і більше [25].

4.3 Проведення інструктажів з охорони праці

Інструктажі з охорони праці відіграють ключову роль у забезпеченні безпеки та здоров'я працівників на робочих місцях. Вони спрямовані на попередження нещасних випадків, виробничих травм та професійних захворювань. Інструктажі допомагають працівникам зрозуміти потенційні

ризики, пов'язані з їхньою роботою, і навчитися діям, які можуть мінімізувати ці ризики.

Існують різні види інструктажів, кожен з яких має своє призначення та особливості проведення. Вступний інструктаж проводиться для всіх нових працівників перед початком їхньої трудової діяльності. Він охоплює загальні питання охорони праці, правила поведінки на робочому місці та порядок дій у разі надзвичайних ситуацій [26].

Первинний інструктаж на робочому місці проводиться безпосередньо перед початком виконання працівником своїх обов'язків. Його мета – ознайомити працівника з конкретними умовами праці, обладнанням, з яким він буде працювати, а також з можливими небезпеками та заходами для їх запобігання. Це особливо важливо для робіт, які пов'язані з підвищеним ризиком або використанням складного обладнання.

Повторний інструктаж проводиться регулярно для всіх працівників незалежно від їхнього стажу роботи та професійної підготовки. Його метою є закріплення знань з охорони праці, а також ознайомлення з новими правилами, стандартами чи змінами у виробничому процесі. Це допомагає підтримувати високий рівень безпеки на робочих місцях і попереджувати випадки порушення правил охорони праці.

Позаплановий інструктаж проводиться у випадках, коли в організації сталися нещасні випадки, аварії або при виявленні порушень правил охорони праці. Такий інструктаж спрямований на аналіз причин інцидентів та запобігання їх повторенню в майбутньому. Він може проводитися також при введенні в дію нових нормативних документів або зміні виробничих процесів [26].

Процес організації інструктажів з охорони праці включає кілька етапів. Спочатку визначається перелік посад, для яких необхідне проведення інструктажів, а також періодичність їх проведення. Далі розробляється програма інструктажів, яка містить необхідні теми та матеріали для навчання.

Підготовка інструкторів або відповідальних осіб, які будуть проводити інструктажі, також є важливим етапом. Інструктори повинні мати відповідну кваліфікацію та знання з охорони праці.

Під час проведення інструктажу використовується різноманітний навчальний матеріал, зокрема презентації, відеоролики, інструкції та наочні матеріали. Важливо, щоб інформація була подана доступно та зрозуміло для всіх працівників незалежно від їхнього рівня освіти та досвіду.

Після проведення інструктажу обов'язково проводиться перевірка знань працівників. Це може бути усне опитування, тестування або практичні завдання. Мета перевірки – переконатися, що працівники правильно засвоїли інформацію та можуть застосовувати її на практиці. Результати перевірки фіксуються в журналі обліку інструктажів.

Ефективне проведення інструктажів з охорони праці сприяє зниженню рівня виробничого травматизму та професійних захворювань. Інструктажі підвищують обізнаність працівників про небезпеки на робочому місці та способи їх уникнення. Вони також сприяють формуванню відповідального ставлення до дотримання правил безпеки, що є важливим аспектом профілактики нещасних випадків.

Крім того, інструктажі допомагають підвищити ефективність роботи, оскільки працівники краще розуміють свої обов'язки та знають, як безпечно виконувати свої завдання. Вони також можуть зменшити кількість простоїв та порушень виробничого процесу, пов'язаних з нещасними випадками або аваріями [26].

Згідно типових положень про порядок проведення навчання та перевірку знань з питань охорони праці НПАОП 0.00-4.36-05. (із змінами від 30.01.2017 р., № 140).

ВИСНОВКИ

Результатом цієї кваліфікаційної робити була розроблена комп'ютеризована система збору та логування показників сенсорів для пожежної сигналізації яка надає можливість моніторингу та аналізу даних з сенсорів температури, вологості та наявності диму. Розроблений веб-інтерфейс для перегляду даних з сенсорів як і в реальному часі так і з бази даних. На веб-сторінці також малюються графіки відносно значень сенсорів та є можливість відключити сенсори для кращого енергозбереження системи. Графічне представлення даних про стан сенсорів полегшує розуміння їх поведінки та змін у часі.

Сенсори забезпечують запис даних про навколишнє середовище які в подальшому передаються на веб-сторінку. Усі зібрані дані автоматично зберігаються у базі даних. Це забезпечує надійне зберігання історії показників сенсорів, що дозволяє аналізувати події та переглядати дані певної дати та години.

Зручний веб-інтерфейс забезпечує користувачам легкий доступ до оперативної інформації, а також можливість вибору даних за конкретні періоди. Цей інтерфейс також дозволяє віддалено керувати включенням та виключенням системи для зменшення енергоспоживання.

В процесі розробки було обґрунтовано вибір апаратних компонентів, включаючи сенсори та мікроконтролери, які найкраще підходять для цієї системи. Проведено детальний аналіз їх характеристик і можливих альтернатив.

Розроблена система відповідає сучасним вимогам до безпеки та інновацій, забезпечуючи комплексний підхід до моніторингу, аналізу і управління даними пожежної сигналізації. Вона може служити базою для подальшого розвитку і вдосконалення систем у цій сфері.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. E-learning TNTU. URL: https://dl.tntu.edu.ua/ (дата звернення: 24.04.24).

2. Осухівська Г. М., Тиш Є. В., Луцик Н. С., Паламар А. М. Методичні вказівки до виконання кваліфікаційних робіт здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль, ТНТУ. 2022. 28 с.

3. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп'ютерні мережі. Книга 1 [навчальний посібник]. Львів : «Магнолія 2006», 2013. 256 с.

4. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп'ютерні мережі. Книга 2. [навчальний посібник]. Львів : "Магнолія 2006", 2014. 312 с.

5. Лупенко С. А., Пасічник В. В., Тиш Є. В. Комп'ютерна логіка. Львів: Видавництво «Магнолія - 2006». 2015. 354 с.

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

7. Вікіпедія. Пожежна сигналізація. URL: https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B6%D0%B5%D0%B6%D0%BD%D0%B0_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F (дата звернення: 25.05.24).

8. Протипожежна сигналізація і системи оповіщення. URL: https://euroservis.com.ua/ua/protivopozharnaya-signalizatsiya-i-sistemy-opoveshcheniya/ (дата звернення: 25.05.24).

9. Datasheet мікроконтролера ESP8266. URL: https://www.espressif.com /sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf (дата звернення: 27.04.24).

10. Розпіновка плати ESP8266. URL: https://www.theengineeringprojects. com/2018/08/esp8266-pinout-datasheet-features-applications.html (дата звернення: 28.04.24).

11. Основні характеристики сенсора газу MQ2. URL: https://www.winsen-sensor.com/product/mq-2.html (дата звернення: 29.04.24).

12. Datasheet сенсора газу MQ2. URL: https://www.pololu.com/file /0J309/MQ2.pdf (дата звернення: 29.04.24).

13. Підключення ESP8266 до сенсора газу MQ2. URL: https://newbiely.com /tutorials/esp8266/esp8266-gas-sensor (дата звернення: 30.04.24).

14. Datasheet сенсора температури та вологості DHT11. URL: https://www. mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf (дата звернення: 01.05.24).

15. Підключення сенсора температури та вологості DHT11 до ESP8266. URL: https://newbiely.com/tutorials/esp8266/esp8266-dht11 (дата звернення: 02.05.24).

16. Базові налаштування та інструкція до програмного забезпечення Arduino IDE. URL: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/ (дата звернення: 13.05.24).

17. Базові налаштування та інструкція до програмного забезпеченняWebStorm. URL: https://www.jetbrains.com/webstorm/features/ (дата звернення: 13.05.24).

18. Встановлення ESP8266 в Arduino IDE. URL: https://randomnerdtutorials. com/how-to-install-esp8266-board-arduino-ide/ (дата звернення: 13.05.24).

19. Підключення ESP8266 до мережі Wi-Fi. URL: https://www.instructables. com/IoT-ESP8266-Series-1-Connect-to-WIFI-Router/ (дата звернення: 14.05.24).

20. Web Server на мікроконтролері ESP8266. URL: https:// randomnerdtutorials.com/esp32-web-server-arduino-ide/ (дата звернення: 14.05.24).

21. Довідка бібліотеки Arduino Json. URL: https://arduinojson.org/v7/tutorial /serialization/ (дата звернення: 15.05.24).

22. ESP8266 NodeMCU HTTP GET and HTTP POST with Arduino IDE. URL: https://randomnerdtutorials.com/esp8266-nodemcu-http-get-post-arduino/ (дата звернення: 16.05.24).

23. Зберігання даних в базу даних. URL: https://esp32.com /viewtopic.php?t=15682 (дата звернення: 17.05.24).

24. Надзвичайні ситуації та їх класифікація. URL: https://osvita.ua /vnz/reports/bjd/22895/ (дата звернення: 22.05.24).

25. Розрахунок кількісті вогнегасників для приміщень. URL: https:// euroservis.com.ua/ua/kak-rasschitat-kolichestvo-ognetushiteley-dlya-pomeshcheniya/ (дата звернення: 23.05.24).

26. Типове положення про порядок проведення навчання та перевірку знань з питань охорони праці. (від 26.01.2005 р., № 15, із змінами від 30.01.2017 р., № 140 – НПАОП 0.00-4.36-05).

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра комп'ютерних систем та мереж

«ЗАТВЕРДЖУЮ»

Завідувач кафедри КС

Осухівська Г.М.

"__"_____2024р.

КОМП'ЮТЕРИЗОВАНА СИСТЕМА ЗБОРУ ТА ЛОГУВАННЯ ПОКАЗНИКІВ СЕНСОРІВ ДЛЯ ПОЖЕЖНОЇ СИГНАЛІЗАЦІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на 7 листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп'ютерна інженерія»

«УЗГОДЖЕНО»
Керівник кваліфікаційної роботи
к.т.н., доцент Яцишин В. В.
"____" _____ 2024 р.

«ВИКОНАВЕЦЬ»
Студент групи СІ-41
Гаврада Д. М.
"____" _____ 2024 р.

Тернопіль 2024

1. Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи бакалавра: «Комп’ютеризована система збору та логування показників сенсорі для пожежної сигналізації»

Умовне позначення дипломного проекту: КС КРБ 123.112.00.00.

1.2 Виконавець

Студент групи СІ-41, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Гаврада Дмитро Миколайович.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи бакалавра є наказ по університету № 4/7-408 від «24» квітня 2024 року

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи бакалавра – 24.04.2023 р.

Плановий термін завершення виконання кваліфікаційної роботи бакалавра –

1.5 Порядок оформлення та пред’явлення результатів роботи

Оформлення технічної документації до кваліфікаційної роботи бакалавра

здійснюється згідно діючих вимог вітчизняних та міжнародних стандартів. Технічна документація до кваліфікаційної роботи бакалавра включає в себе текст пояснювальної записки та креслення, які максимально інформативно та стисло відображають основні результати розробки комп’ютеризованої системи збору та логування показників сенсорі для пожежної сигналізації.

Основними регламентними документами при оформленні та пред’явлені результатів проектування є групи діючих стандартів ДСТУ, ГОСТ, ISO та ЄСКД, ЕСПД.

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

2 Призначення і цілі створення системи

2.1 Призначення системи

Система призначена для збору та логування показників сенсорів для пожежної сигналізації, для подальшого збереження та відображення на веб-сторінці.

2.2 Мета створення системи

Метою цієї кваліфікаційної роботи є розробка комп'ютеризованої системи збору та логування показників сенсорів пожежної сигналізації, яка буде збирати та надсилати дані на веб-сторінку з сенсорів в режимі реального часу, зберігати дані в базі даних для подальшого аналізу та візуалізувати дані про стан сенсорів в вигляді графіків.

2.3 Характеристика об'єкту

Система проектується для збору та логування показників сенсорів з пожежної сигналізації, що включає в себе:

– розробку структурної схеми;

– розробку схеми електричної принципової;

– розробку алгоритму роботи, розробку програмного забезпечення для мікроконтролера та розробку веб-сторінки.

3 Вимоги до системи

3.1 Вимоги до системи в цілому

Комп’ютеризована система збору та логування показників сенсорів для пожежної сигналізації повинна забезпечити:

1. Достовірність даних які передаються;
2. Невелику затримку передачі даних;
3. Виведення даних в реальному часі, відмальовування графіків та зберігання даних в базу даних;
4. Оптимізацію роботи системи.

3.1.1 Вимоги до структури та функціонування системи

Структура системи збору та логування показників сенсорів для пожежної сигналізації включає в себе:

– мікроконтролер, який забезпечить загальне керування функціонуванням системи;

– сенсор газу;

– сенсор температури та вологості;

– світлодіоди;

– резистори.

В загальному випадку, структура системи повинна реалізовувати функції

збору та логування показників сенсорів для пожежної сигналізації. Основні функціональні вимоги характеризуються наступними критеріями:

– функціональність;
– точність;
– ефективність;
– надійність;
– сумісність.

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

Обмін даними між контролером та комп’ютером повинен здійснюватися з використанням бездротових технологій передачі інформації.

3.1.3 Вимоги до надійності системи

Система повинна бути захищена від фізичних чи механічних пошкоджень на рівні апаратного та програмного забезпечення. Надійність системи повинна забезпечувати відновлюваність функціонування у випадку збою апаратного чи програмного забезпечення.

3.1.7 Вимоги до апаратного забезпечення

Вимоги до елементної бази розробки:

– режими роботи і умови експлуатації вибраних елементів повинні відповідати вказаним в ТЗ;
– елементна база по можливості має бути широковживаною, доступною і дешевою. Необхідно також враховувати можливість заміни вибраних елементів на аналогічні (вітчизняні чи імпортного виробництва).

Вимоги до мікроконтролера:

– мікроконтролер має підтримувати RISC архітектуру команд;

– мікроконтролер повинен містити необхідний набір вбудованих периферійних пристроїв (таймери, АЦП і т.п.) та потрібну кількість керованих портів введення /виведення.

– наявність вбудованого Wi-Fi модуля.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ.

Комплект конструкторської документації повинен складатись з:

– пояснювальної записки;

– графічного матеріалу:

1. структурна схема;
2. схема електрична принципова;
3. блок-схема алгоритму програми;
4. блок-схема веб-сторіки.

*Примітка: В комплект конструкторської документації можуть вноситися зміни та доповнення в процесі розробки.

5 Техніко-економічні показники

Собівартість розробки системи повинна становити не більше 1500 грн.

Термін експлуатації системи повинен бути не менший 10 років.

*Примітка: собівартість системи може змінюватись під час розрахунку в процесі розробки.

6 Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання КРБ

№ етапу	Назва етапу виконання КРБ	Термін виконання
1	Отримання індивідуального завдання	01.02 – 09.02
2	Аналіз отриманого завдання	05.02 – 11.02
3	Ознайомлення з документацією мікроконтролера ESP8266	27.04.24 – 28.04.24
4	Ознайомлення з документацією сенсора MQ2	29.04 – 30.04
5	Ознайомлення з документацією сенсора DHT11	01.05 – 02.05
6	Ознайомлення з бібліотеками створення WebServer на мікроконтролері ESP8266	03.05 – 04.05
7	Ознайомлення з бібліотеками перетворення даних з сенсорів для передавання в базу даних та на створений WebServer на мікроконтролері ESP8266	05.05.24 – 06.05.24
8	Створення електрично-принципової схеми пристрою	07.05 – 08.05
9	Проектування пристрою за схемою	09.05 – 10.05
10	Написання програмного забезпечення	13.05 – 19.05
11	Тестування програмного забезпечення	20.05.24
12	Тестування створеного пристрою	21.05.24
13	Безпека життєдіяльностіі, основи охорони праці	10.06 – 15.06
14	Оформлення кваліфікаційної роботи	16.06 – 20.06
15	Попередній захист кваліфікаційної роботи	14.06
16	Захист кваліфікаційної роботи	24.06

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

Додаток Б

Перелік елементів

Позн.	Найменування	К-сть	Примітка
	Світлодіоди		
D1	HB3b-245R	1	
D1	500G1D-YETDA	1	
	Резистори		
R1, R2	Murata El. 10k	2	
	Плата		
U1	ESP8266 NodeMCU	1	
	Сенсори		
U2	DHT11	1	
U3	MQ2	1	

Зм.	Арк.	№ документа	Підпис	Дата	КС КРБ 123.112.00.00 ПЕ		
Розробив		Гаврада Д.М.			Комп'ютеризована система збору та логування показників сенсорів для пожежної сигналізації Перелік елементів	Літ	Аркуш 79 / Аркушів 1
Перевірив		Яцишин В.В.				н	
Консульт.						ТНТУ, каф. КС, гр. СІ-41	
Н. контр.		Тиш Є.В.					
Зав. каф.		Осухівська Г.М.					

Додаток В

Лістинг коду мікроконтролера

```
//#define IS_DEV

// Import required libraries
#include "ESP8266WiFi.h"
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>
#include <DHT.h>
#include "AsyncJson.h"
#include "ArduinoJson.h"
#include <FS.h>
#include <stdbool.h>
#include <Vector.h>

#ifdef IS_DEV
  #include <Streaming.h>
#endif

//PIN Defines
#define MQ_PIN A0

#define DHT_PIN D6

#define GREEN_PIN D5
#define RED_PIN D2
//PIN Defines END

//WIFI
const char* ssid = "LiLSaiko";
const char* password = "1q2a3z4e5d";
//WIFI END

//Storage
const int ELEMENT_COUNT_MAX = 33;
bool MQ_ARRAY[ELEMENT_COUNT_MAX];
Vector<bool> MQ_Vector(MQ_ARRAY);

int T_ARRAY[ELEMENT_COUNT_MAX];
Vector<int> T_Vector(T_ARRAY);

int H_ARRAY[ELEMENT_COUNT_MAX];
Vector<int> H_Vector(H_ARRAY);

bool state = false;

bool needToUpdateData = false;
String payload;
String payloadUpdatedAt;
String timeStamp;
```

```
//Storage END

//Server
AsyncWebServer server(80);
String goodServerIp;

#define DHT_TYPE DHT11

DHT dht(DHT_PIN, DHT_TYPE);

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);

  pinMode(GREEN_PIN, OUTPUT);
  pinMode(RED_PIN, OUTPUT);

  digitalWrite(GREEN_PIN, HIGH);
  digitalWrite(RED_PIN, HIGH);
  dht.begin();

  // Initialize SPIFFS
  if(!SPIFFS.begin()) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());

  while (true) {
    goodServerIp = Serial.readStringUntil('\n');
    digitalWrite(RED_PIN, !digitalRead(RED_PIN));
    if (goodServerIp == "") continue;
    if (checkConnection(goodServerIp)) {
      Serial.println("Connected");
      digitalWrite(RED_PIN, HIGH);
      break;
    } else {
      Serial.println("Error");
    }
  }

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send(SPIFFS, "/index.html", String());
  });
```

```
  server.on("/Graphics.html", HTTP_GET, [](AsyncWebServerRequest
*request) {
    request->send(SPIFFS, "/Graphics.html", String());
  });

  server.on("/Sensors.html", HTTP_GET, [](AsyncWebServerRequest
*request) {
    request->send(SPIFFS, "/Sensors.html", String());
  });

  server.on("/Database.html", HTTP_GET, [](AsyncWebServerRequest
*request) {
    request->send(SPIFFS, "/Database.html", String());
  });

  // Route to load style.css file
  server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest
*request) {
    request->send(SPIFFS, "/style.css", "text/css");
  });

  server.on("/data", HTTP_GET, [](AsyncWebServerRequest *request)
{
    AsyncResponseStream *response = request-
>beginResponseStream("application/json");
    JsonDocument root;
    JsonObject obj = root.to<JsonObject>();

    obj["curState"] = state;

    JsonArray MQArr = obj.createNestedArray("mqData");
    for (int element : MQ_Vector) {
      MQArr.add(element);
    }

    JsonArray TArr = obj.createNestedArray("tData");
    for (int element : T_Vector) {
      TArr.add(element);
    }

    JsonArray HArr = obj.createNestedArray("hData");
    for (int element : H_Vector) {
      HArr.add(element);
    }

    serializeJson(root, *response);
    request->send(response);
  });

  server.on("/getdata", HTTP_GET, [](AsyncWebServerRequest
*request) {
    AsyncResponseStream *response = request-
>beginResponseStream("application/json");

    JsonDocument doc;
```

```
    deserializeJson(doc, payload);
    doc["updateAt"] = payloadUpdatedAt;

    serializeJson(doc, *response);
    request->send(response);
  });

  server.on("/switch", HTTP_POST, [](AsyncWebServerRequest
*request) {
      state = !state;
      AsyncResponseStream *response = request-
>beginResponseStream("application/json");

      JsonDocument root;
      root["state"] = state;

      serializeJson(root, *response);
      request->send(response);
  });

  server.on("/db", HTTP_GET, [](AsyncWebServerRequest *request) {
    timeStamp = request->getParam(0)->value();
    needToUpdateData = true;
    request->send(200);
  });

  // Start server
  server.begin();
}

void loop() {
  if (needToUpdateData) {
    updateData();

    needToUpdateData = false;
  }

  if (state) {
    digitalWrite(GREEN_PIN, LOW);
    digitalWrite(RED_PIN, HIGH);

    removeElements();

    int temp = readT();
    int hum = readH();
    bool gas = readMQ();

    sendDataToServer(temp, hum, gas);
  } else {
    digitalWrite(RED_PIN, LOW);
    digitalWrite(GREEN_PIN, HIGH);
  }

  delay(1000);
}
```

```
void removeElements() {
  while (MQ_Vector.size() > ELEMENT_COUNT_MAX - 1) {
    MQ_Vector.remove(0);
  }

  while (T_Vector.size() > ELEMENT_COUNT_MAX - 1) {
    T_Vector.remove(0);
  }

  while (H_Vector.size() > ELEMENT_COUNT_MAX - 1) {
    H_Vector.remove(0);
  }
}

bool readMQ() {
  bool gasValue = analogRead(MQ_PIN)>180;

  Serial.print("MQ2 sensor AO value: ");
  Serial.println(gasValue);

  MQ_Vector.push_back(gasValue);
  return gasValue;
}

int readT() {
  int temp = dht.readTemperature();

  T_Vector.push_back(temp);
  return temp;
}

int readH() {
  int hum = dht.readHumidity();

  H_Vector.push_back(hum);
  return hum;
}

bool checkConnection(String ip) {
  bool askStatus = false;

  if (WiFi.status() == WL_CONNECTED) {
      WiFiClient client;
      HTTPClient http;

      String serverPath = "http://" + ip + "/check";

      http.begin(client, serverPath.c_str());

      int httpResponseCode = http.GET();

      if (httpResponseCode > 0) {
        String payload = http.getString();
```

```
        JsonDocument doc;
        deserializeJson(doc, payload);
        if (doc["server"] == "esp32") askStatus = true;
      } else {
        Serial.print("Error code: ");
        Serial.println(httpResponseCode);
        askStatus = false;
      }
      http.end();
  }
  return askStatus;
}

void sendDataToServer(int temp, int hum, bool gas) {
   WiFiClient client;
   HTTPClient http;

   char output[128];
   JsonDocument doc;
   doc["temp"] = temp;
   doc["hum"] = hum;
   doc["gas"] = gas;

   http.begin(client, "http://" + goodServerIp + "/data");
   http.addHeader("Content-Type", "application/json");

   serializeJson(doc, output);
   int httpResponseCode = http.POST(output);

   Serial.print("HTTP Response code: ");
   Serial.println(httpResponseCode);

   http.end();
}

void updateData() {
    WiFiClient client;
    HTTPClient http;

    String serverPath = "http://" + goodServerIp + "/data?date=" +
timeStamp;
    Serial.print(serverPath);
    http.begin(client, serverPath.c_str());

    int httpResponseCode = http.GET();

    if (httpResponseCode > 0) {
      payload = http.getString();
      payloadUpdatedAt = timeStamp;
    } else {
      Serial.print("Something went wrong");
    }

    http.end();
}
```

Додаток Д

Лістинг основних частин коду веб-інтерфейсу

Лістинг основної HTML-сторінки:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="style.css">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"
            integrity="sha256-
/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>
    <title>ESP8266 Web Server</title>
</head>

<body>

<header>
    <p>ПОЖЕЖНА СИГНАЛІЗАЦІЯ</p>
    <div class="buttons">
        <button
onclick="window.location.href='/'">Головна</button>
        <button
onclick="window.location.href='Sensors.html'">Показники</button>
        <button
onclick="window.location.href='Graphics.html'">Графіки</button>
        <button onclick="window.location.href='Database.html'"
style="width: 120px">База даних</button>
    </div>
    <div style="flex: 1">

    </div>
</header>

<div class="main-content" style="height: 90%">
    <button class="mainButton" id="switch-button"
onclick="switchState()">...</button>
</div>

<footer>
    <p>SYSTEM STATUS: <a id="state"></a></p>
</footer>

<script>
    const changeState = (state) => {
        $('#switch-button').removeClass(['buttonOff',
'buttonOn']);
```

```
        if (state) {
            $('#switch-button').addClass('buttonOff');
            $('#switch-button').html('OFF');

            $('#state').css("color", "green");
            $('#state').html('ON');
        } else {
            $('#switch-button').addClass('buttonOn');
            $('#switch-button').html('ON');

            $('#state').css("color", "red");
            $('#state').html('OFF');
        }
    }

    $.get("/data", function (data) {
       changeState(data.curState);
    });

    let isRunning = false;

    const switchState = () => {
        if (isRunning) return;
        isRunning = true;
        $.post("/switch", function (data) {
            changeState(data.state);

            isRunning = false;
        })
    }
</script>

</body>
</html>
```

Лістинг HTML-сторінки з графіками:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="style.css">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"
            integrity="sha256-
/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>
    <title>ESP8266 Web Server</title>
</head>

<body>
```

```
<header>
    <p>ПОЖЕЖНА СИГНАЛІЗАЦІЯ</p>
    <div class="buttons">
        <button
onclick="window.location.href='/'">Головна</button>
        <button
onclick="window.location.href='Sensors.html'">Показники</button>
        <button
onclick="window.location.href='Graphics.html'">Графіки</button>
        <button onclick="window.location.href='Database.html'"
style="width: 120px">База даних</button>
    </div>
    <div style="flex: 1"></div>

</header>

<div class="main-content" style="height: 90%; gap: 30px">
    <div>
        <p class="graphText">Temperature</p>
        <div class="graphBox">
            <div class="coordinates">
                <p>50</p>
                <p>40</p>
                <p>30</p>
                <p>20</p>
                <p>10</p>
            </div>
            <div>
                <canvas id="temp" height="500"
width="500"></canvas>
                <div class="coordinatesBtm">
                    <p>5</p>
                    <p>10</p>
                    <p>15</p>
                    <p>20</p>
                    <p>25</p>
                    <p>30</p>
                </div>
            </div>
        </div>
    </div>

    <div>
        <p class="graphText">Humidity</p>
        <div class="graphBox">
            <div class="coordinates">
                <p>100%</p>
                <p>90%</p>
                <p>80%</p>
                <p>70%</p>
                <p>60%</p>
                <p>50%</p>
                <p>40%</p>
                <p>30%</p>
```

```
                <p>20%</p>
                <p>10%</p>
            </div>
            <div>
                <canvas id="hum" height="500"
width="500"></canvas>

                <div class="coordinatesBtm">
                    <p>5</p>
                    <p>10</p>
                    <p>15</p>
                    <p>20</p>
                    <p>25</p>
                    <p>30</p>
                </div>
            </div>
        </div>
    </div>
</div>

<footer>
    <p>SYSTEM STATUS: <a id="state"></a></p>
</footer>

<script>
    const changeState = (state) => {
        $('#switch-button').removeClass(['buttonOff',
'buttonOn']);

        if (state) {
            $('#switch-button').addClass('buttonOff');
            $('#switch-button').html('OFF');

            $('#state').css("color", "green");
            $('#state').html('ON');
        } else {
            $('#switch-button').addClass('buttonOn');
            $('#switch-button').html('ON');

            $('#state').css("color", "red");
            $('#state').html('OFF');
        }
    }

    const TValues = [];
    const HValues = [];

    const TC = document.getElementById("temp");
    const TCtx = TC.getContext("2d");

    const HC = document.getElementById("hum");
    const HCtx = HC.getContext("2d");

    const draw = () => {
        TCtx.reset();
```

```
        TCtx.strokeStyle = 'white';
        TCtx.lineWidth = '2';
        TCtx.fillText("0", 0, 500);
        TCtx.moveTo(0, 500 - TValues[0]);
        TValues.forEach((el, idx) => {
            TCtx.lineTo(idx * 15, 500 - (el * 10));
        })
        TCtx.stroke();

        HCtx.reset();
        HCtx.strokeStyle = 'white';
        HCtx.lineWidth = '2';
        HCtx.moveTo(0, 500 - HValues[0]);
        HValues.forEach((el, idx) => {
            HCtx.lineTo(idx * 15, 500 - (el * 5));
        })
        HCtx.stroke();
    }

    draw();

    makeMagic();

    setInterval(() => {
        makeMagic();
    }, 1000);
</script>
</body>
</html>



<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="style.css">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"
            integrity="sha256-
/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>
    <title>ESP8266 Web Server</title>
</head>

<body>

<header>
    <p>ПОЖЕЖНА СИГНАЛІЗАЦІЯ</p>
    <div class="buttons">
        <button
onclick="window.location.href='/'">Головна</button>
```

```
        <button
onclick="window.location.href='Sensors.html'">Показники</button>
        <button
onclick="window.location.href='Graphics.html'">Графіки</button>
        <button onclick="window.location.href='Database.html'"
style="width: 120px">База даних</button>
    </div>
    <div style="flex: 1"></div>
</header>


<div class="main-content" style="flex-direction: column; height:
90%">
    <div style="display: flex; flex-direction: column">
        <div class="inputData">
            <p>
               Виберіть дату щоб продовжити:
            </p>
            <input
                    type="datetime-local"
                    id="databaseDateTime"
                    value="2024-05-20T19:30"/>
        </div>
        <table class="table">
            <thead>
            <tr>
                <td class="column">Temperature</td>
                <td class="column">Humidity</td>
                <td class="column">Presence of gas</td>
                <td style="width: 16px; padding: 0"></td>
            </tr>
            </thead>
            <tbody id="tableBody">
            </tbody>
        </table>
    </div>
</div>

<footer>
    <p>SYSTEM STATUS: <a id="state"></a></p>
</footer>

<script>
    const changeState = (state) => {
        $('#switch-button').removeClass(['buttonOff',
'buttonOn']);

        if (state) {
            $('#switch-button').addClass('buttonOff');
            $('#switch-button').html('OFF');

            $('#state').css("color", "green");
            $('#state').html('ON');
        } else {
            $('#switch-button').addClass('buttonOn');
```

```
            $('#switch-button').html('ON');

            $('#state').css("color", "red");
            $('#state').html('OFF');
        }
    }

    $("#databaseDateTime").on( "change", function() {
        const value = $("#databaseDateTime").val();

        $.get(`/db?t=${value}`, function (data) {
            setTimeout(makeMagic, 1000);
        });
    } );

    const fillTable = (data) => {
        $("#tableBody").empty()

        const table = document.getElementById('tableBody');

        for (let i = 0; i < data.length; i++) {
            const currentData = data[i];

            const row = table.insertRow();

            const cell1 = row.insertCell(0);
            const cell2 = row.insertCell(1);
            const cell3 = row.insertCell(2);

            cell1.innerHTML = currentData.temperature;
            cell2.innerHTML = currentData.humidity;
            cell3.innerHTML = !!currentData.gaz ? "Yes" : "No";
        }
    }

    const makeMagic = () => {
        $.get("/getdata", function (data) {
            console.log(data);
            const value = $("#databaseDateTime").val();

            if (value === data.updateAt) {
                fillTable(data.result);
            } else {
                setTimeout(makeMagic, 1000);
            }
        });
    }

    setInterval(() => {
        $.get("/data", function (data) {
            changeState(data.curState);
        });
    }, 1000);

</script>
```

```
</body>
</html>
```

Лістинг CSS коду:

```
@import
url('https://fonts.googleapis.com/css2?family=Lato:ital,wght@0,100
;0,300;0,400;0,700;0,900;1,100;1,300;1,400;1,700;1,900&display=swa
p');

html, body {
  height: 100%;
  width: 100%;
  margin: 0;
  padding: 0;
}

body {
  background-image: url("https://i.imgur.com/erPWads.jpeg");

  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
}

header {
  display: flex;
  flex-direction: row;
  width: 100%;

  margin: 0;
}

header p {
  font-size: 30px;
  font-weight: bold;
  flex: 1;

  color: #6F8CC3;

  font-family: "Lato", sans-serif;
  align-self: center;

  margin: 0 0 0 20px;
}

.buttons {
  display: flex;
  flex: 1;

  justify-content: center;
  column-gap: 50px;

  padding: 20px;
}
```

```
.buttons button {
  background: transparent;
  border-color: transparent;
  border-bottom-width: 3px;
  border-left-width: 0;
  border-right-width: 0;
  border-top-width: 0;

  color: #6F8CC3;
  font-size: 20px;

  font-family: "Lato", sans-serif;

  transition-duration: .2s;
}

.buttons button:hover {
  color: #E6EAF1;
  border-color: #E6EAF1;

  border-bottom-width: 3px;
}

.main-content {
  display: flex;
  flex: 1;

  height: 100%;
  width: 100%;

  margin: 0;
  padding: 0;

  align-items: center;
  justify-content: center;
}

.boxButton {
  display: flex;

  height: 100%;
  width: 100%;

  box-sizing: border-box;

  align-items: center;
  justify-content: center;
}

.mainButton {
  height: 300px;
  width: 300px;
  border-radius: 50%;
  font-size: 70px;
```

```
  border-width: 20px;
  border-style: solid;
}

.buttonOn {
  color: #8FA3C9;
  background-color: #567694;
  border-color: #364365;
  box-shadow: 0 0 200px #8FA3C9;
}

.buttonOff {
  color: #3A1E1E;
  background-color: #581C1C;
  border-color: #2D2424;
  box-shadow: 0 0 200px #581C1C;
}

table {
  width: 700px;
  height: 600px;

  color: white;

  background-color: rgba(14, 18, 29, 0.9);

  border: 1px solid white;
  border-collapse: collapse;

  box-shadow: 0 0 50px white;
}

tr, td {
  color: white;
  font-family: "Lato", sans-serif;

  text-align: center;
  border: 1px solid white;
}

thead td {
  font-size: 25px;

  padding: 10px;
}

tbody td {
  font-size: 20px;
}

.graphBox {
  display: flex;
  flex-direction: row;
}
```

```
.graphText {
  text-align: center;

  margin-bottom: 0;

  color: white;

  font-size: 25px;
  font-weight: bold;

}

.coordinates {
  display: flex;
  flex-direction: column;
  justify-content: space-between;

  padding: 20px 0 35px 0;
}

.coordinates p {
  color: white;
  font-size: 20px;

  margin: 0;
  padding: 0;

  text-align: right;
}

.coordinatesBtm {
  display: flex;
  flex-direction: row;
  justify-content: space-between;

  padding: 0 20px 0 20px;
}

.coordinatesBtm p {
  color: white;
  font-size: 20px;

  margin: 0;
  padding: 0;
}


canvas {
  margin: 20px 20px 5px 10px;

  max-height: 500px;
  max-width: 500px;

  border: 3px solid #ffffff;
  background-color: rgba(14, 18, 29, 0.9);
```

```
  box-shadow: 0 0 50px white;

  display: flex;
  justify-content: space-between;
}

.inputData {
  display: flex;
  justify-content: space-between;
  padding: 0 0 10px 5px;
}

.inputData p {
  display: flex;
  align-self: center;

  color: white;
  font-size: 25px;

  margin: 0;

  text-shadow: 0 0 30px white;

  font-family: "Lato", sans-serif;
}

.inputData input {
  color: white;
  font-size: 20px;

  font-family: "Lato", sans-serif;

  width: 200px;
  padding: 10px;

  color-scheme: dark;

  border: 1px solid #ffffff;
  background-color: rgba(14, 18, 29, 0.9);

  box-shadow: 0 0 50px white;
}

footer {
  display: flex;

  text-align: center;
  justify-content: center;
}

footer p {
  font-size: 20px;
  font-weight: bold;
  color: #A4B9E1;
```

```
  margin: 0 auto;
}

.table{
  table-layout: fixed;

}

.table tbody{
  display:block;
  width: 698px;
  height: 560px;

  overflow: scroll;
  overflow-x: hidden;
  scrollbar-color: white transparent;
  }


.table tbody th, .table tbody td {
  height: 50px;
  min-width: 224px;
}

.column {
  min-width: 224px;

}
```