Хмельницький національний університет

Факультет інформаційних технологій

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Тростянецького Назара Олексійовича

на здобуття ступеня вищої освіти Бакалавра

Програмне забезпечення системи моніторингу кліматичних умов та електричних параметрів приміщення (в складі комплексного дипломного проєкту “Система моніторингу кліматичних та електричних параметрів приміщення”)

Галузь знань 12 – Інформаційні технології

Спеціальність 123 – Комп’ютерна інженерія

Освітня програма Програмування та захист комп`ютерних систем і мереж

Шифр КРБКІ.2001123.20.01.06 ПЗ

Виконав студент 4 курсу група КІ1-20-1 Назар ТРОСТЯНЕЦЬКИЙ

Керівник канд. техн. наук, доцент Юрій КЛЬОЦ

Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

19 06 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 123 – Комп'ютерна інженерія

Освітня програма Програмування та захист комп`ютерних систем і мереж

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Тростянецькому Назару Олексійовичу

1 Тема роботи Програмне забезпечення системи моніторингу кліматичних умов та електричних параметрів приміщення (в складі комплексного дипломного проєкту "Система моніторингу кліматичних та електричних параметрів приміщення")

Керівник роботи к.т.н., доц. Кльоц Ю.П.

Затверджено наказом ректора університету від 15 лютого 2024 № 8

2 Строк подання студентом кваліфікаційної роботи на кафедру 1.06.2024р.

3 Вихідні дані до роботи Реалізувати програмне забезпечення системи моніторингу кліматичних умов та електричних параметрів приміщення.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вибір середовища розробки. Вибір бібліотек для розробки. Написання коду програми. Розроблення схем алгоритмів роботи програми. Опис розробленого коду. Тестування роботи та налаштування.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схема алгоритму загального; схема алгоритма надсилання даних на Blynk; схема алгоритма зчитування параметрів

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Мостовий С.В., старший викладач кафедри кібербезпеки		

7 Дата видачі завдання 16 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	
Ознайомлення з предметною областю	Лютий	
Дослідження існуючих рішень	Лютий	
Постановка задачі	Березень	
Визначення загальних принципів рішення задачі	Березень	
Деталізація принципів рішення задачі	Квітень	
Розробка проєктних рішень	Квітень	
Апробація проєктних рішень	Травень	
Оформлення пояснювальної записки згідно вимог	Травень	
Оформлення графічної частини	Червень	
Захист КР	Червень	

Студент Назар ТРОСТЯНЕЦЬКИЙ

Керівник кваліфікаційної роботи Юрій КЛЬОЦ

№ р я д к а	ф о р м а т	Позначення	Найменування	К і л . л и с т і в	№ е к з	П р и м і т к а
			Текстові документи			
1		КРБКІ.2001123.20.01.06 ПЗ	Пояснювальна записка	80		
			Графічні матеріали			
2		КРБКІ.2001123.20.01.06 Е8	Алгоритм роботи	3		

Зм	Арк	№ докум	Підпис	Дата	КРБКІ.2001123.20.01.06 ВП			
Розробив		Тростянецький Н.О.		13.06.24	Програмне забезпечення системи моніторингу кліматичних умов та електричних параметрів приміщення (в складі комплексного дипломного проєкту "Система моніторингу кліматичних та електричних параметрів приміщення") Відомість проєкту	Літера	Аркуш	Аркушів
Перевір.		Кльоц Ю.П.		18.06.24		У	1	1
Н. контр.		Мостовий С.В.		18.06.24		ХНУ, КІ1–20–1		
Затв.		Кльоц Ю.П.		18.06.24				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмне забезпечення системи моніторингу кліматичних умов та електричних параметрів приміщення (в складі комплексного дипломного проєкту «Система моніторингу кліматичних та електричних параметрів приміщення»)»

Автор роботи: Тростянецький Назар Олексійович.

Керівник роботи: Кльоц Юрій Павлович.

Пояснювальна записка: 80 с., 3 додатка, 18 рис., 42 джерел.

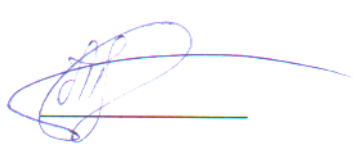
Графічна частина: 3 плакати.

ПЕРЕЛІК КЛЮЧОВИХ СЛІВ: БІБЛІОТЕКА, ПОКАЗНИК, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ARDUINO, BLYNK, ESP32, IDE.

Метою даної дипломної роботи є розробка програмного забезпечення для інтегрованої системи моніторингу параметрів навколишнього середовища на основі мікроконтролера ESP32 та платформи Blynk. Для цієї цілі було проведено аналіз існуючих рішень та технологій для моніторингу параметрів середовища. Вибрано оптимальні компоненти системи, включаючи датчики, мікроконтролер та програмне забезпечення. Розроблено програмне забезпечення для збирання, обробки та відображення даних з датчиків, а також для інтеграції з платформою Blynk.

Розроблена та впроваджена система моніторингу параметрів навколишнього середовища, яка використовує мікроконтролер ESP32 та сервіс Blynk для віддаленого управління та моніторингу даних. Система дозволяє зчитувати дані з датчиків температури, вологості, газу та полум'я, передавати ці дані на сервер Blynk і відображати їх у реальному часі на мобільному додатку, а також здійснювати сповіщення у разі перевищення допустимих значень.

11.06.2024



ABSTRACT

Course project: «Software of the system for monitoring the climate conditions and electrical parameters of the room (in part of the comprehensive diploma project "monitoring system of the climate and electrical parameters of the room")»

Author of the work: Trostyanetskiy Nazar Oleksiyovych.

Supervisor: Klets Yuriy Pavlovich.

Amount: 80 c, 3 appendices, 18 figures, 42 sources.

Graphic part: 3 drawings.

LIST OF KEYWORDS: ARDUINO, BLYNK, LIBRARY, INDICATOR, SOFTWARE , ESP32, IDE.

The purpose of this thesis is to develop software for an integrated system for monitoring environmental parameters based on the ESP32 microcontroller and the Blynk platform. To achieve this goal, an analysis of existing solutions and technologies for monitoring environmental parameters was carried out. Optimal system components including sensors, microcontroller and software are selected. Developed software for collecting, processing and displaying data from sensors, as well as for integration with the Blynk platform.

A system for monitoring environmental parameters was developed and implemented, which uses an ESP32 microcontroller and the Blynk service for remote control and data monitoring. The system allows you to read data from temperature, humidity, gas and flame sensors, transfer these data to the Blynk server and display them in real time on the mobile application, as well as issue notifications in case of exceeding permissible values.

11.06.2024

ЗМІСТ

ВСТУП

У сучасному світі розвиток технологій та зростання попиту на інтелектуальні системи управління викликають необхідність розробки нових рішень для моніторингу параметрів навколишнього середовища. Системи на базі Інтернету речей (IoT) відкривають нові можливості для ефективного управління, забезпечення безпеки та комфорту в різноманітних приміщеннях, таких як житлові будинки, офіси та промислові об'єкти [37].

Завдяки використанню IoT-технологій можна здійснювати віддалений контроль за станом середовища, вчасно реагувати на небезпечні ситуації, наприклад, витік газу або виявлення полум'я, та управляти енергоспоживанням. Це дозволяє не тільки підвищити рівень безпеки, але й знизити експлуатаційні витрати та покращити якість життя.

Метою даної дипломної роботи є розробка інтегрованої системи моніторингу параметрів навколишнього середовища на основі мікроконтролера ESP32 та платформи Blynk [7, 2]. Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Провести аналіз існуючих рішень та технологій для моніторингу параметрів середовища, зокрема тих, що використовують мікроконтролери та IoT-технології.

2. Вибрати оптимальні компоненти системи, включаючи датчики, мікроконтролер та програмне забезпечення, обґрунтувати їх вибір.

3. Розробити програмне забезпечення для збирання, обробки та відображення даних з датчиків, а також для інтеграції з платформою Blynk.

4. Провести тестування розробленої системи в різних умовах експлуатації для оцінки її ефективності та надійності.

5. Порівняти розроблене рішення з існуючими аналогами, визначити його переваги та недоліки.

Сучасні системи моніторингу параметрів середовища використовують різні платформи та мікроконтролери, такі як Arduino, Raspberry Pi та ESP32 [1, 7, 42]. Кожна з цих платформ має свої переваги та недоліки, які враховуються при виборі компонента для конкретної задачі.

Платформа ESP32 відзначається високою продуктивністю, вбудованими модулями Wi-Fi та Bluetooth, а також широкими можливостями для розширення. Використання ESP32 у поєднанні з платформою Blynk дозволяє створювати інтерактивні інтерфейси для моніторингу та управління системою в режимі реального часу.

Програмне забезпечення було розроблене з використанням середовища Arduino IDE. Основні задачі, які вирішувалися під час розробки ПЗ, включають ініціалізацію компонентів системи, збирання даних з датчиків, обробку цих даних та їх передачу на платформу Blynk для подальшого відображення та аналізу. Особлива увага приділялася надійності та стабільності роботи системи, а також можливостям для подальшого розширення функціоналу.

Розроблена система була протестована в реальних умовах експлуатації. Результати тестування показали високу точність та надійність зчитування даних, а також ефективність віддаленого моніторингу через платформу Blynk. Порівняння з іншими рішеннями підтвердило, що використання ESP32 та Blynk забезпечує оптимальний баланс між вартістю, функціональністю та простотою використання.

Таким чином, у дипломній роботі була розроблена ефективна система моніторингу параметрів середовища на основі мікроконтролера ESP32 та платформи Blynk. Розроблене рішення дозволяє здійснювати точний та надійний моніторинг умов середовища, забезпечуючи високу функціональність та зручність у використанні.

1 АНАЛІЗ ЗАСОБІВ РОЗРОБКИ ПРОГРАМИ КЕРУВАННЯ ПРИСТРОЄМ

1.1 Параметри для аналізу

Метою даного проекту є створення приладу для вимірювання температури, вологості, напруги та виявлення наявності вогню. Цей прилад призначений для використання в приміщеннях, в яких необхідно моніторити та забезпечувати безпеку шляхом своєчасного виявлення небезпечних умов, таких як пожежа. Такими приміщеннями можуть бути серверні кімнати, як в нашому випадку. В основі проекту лежить платформа Arduino, яка дозволяє легко інтегрувати різні сенсори та розробити програмне забезпечення для їх управління та обробки даних [1]. Отримані дані з датчиків, будуть виводитись на дисплей прилада, далі за допомогою додатка Blynk, власник може моніторити ситуацію в серверній через свій мобільний телефон або комп'ютер [2]. Дані з пристрою, повинні бути подані максимально просто і зрозуміло, аби власник міг відразу оцінити ситуацію.

Контроль температури повітря в серверній кімнаті є надзвичайно важливим для забезпечення стабільної роботи серверів. Оптимальні температурні умови, за рекомендаціями ASHRAE (American Society of Heating, Refrigerating and Air-Conditioning Engineers) для серверного обладнання зазвичай становлять від 18°C до 27°C [3]. Підвищена температура є небезпечною для компонентів сервера, оскільки це знижує продуктивність та скорочення терміну експлуатації обладнання. Перегрівання серверів може мати такі наслідки, як збої в роботі, короткі замикання, пошкодження комплектуючих, та інші технічні проблеми, які можуть призвести до часткової або повної втрати даних і до збої в роботі послуг. В свою чергу низька температура може впливати на роботу серверного обладнання наступними способами. Через конденсацію вологи при дуже низьких температурах, коли серверна кімната знову нагрівається або коли обладнання працює в холодному середовищі, може відбуватися конденсація

вологи на електронних компонентах. Це може призвести до коротких замикань та пошкоджень обладнання. В'язкість охолоджуючих рідин: низькі температури можуть підвищити в'язкість охолоджуючих рідин або мастил у вентиляторах та інших механічних частинах, що може призвести до їх неправильного функціонування або підвищеного зносу. Стабільність електронних компонентів: деякі електронні компоненти можуть функціонувати менш ефективно або нестабільно при низьких температурах. Наприклад, електролітичні конденсатори можуть втратити свою ємність, що вплине на стабільність електроживлення. Зниження ефективності батарей, якщо серверне обладнання містить резервні батареї або інші джерела живлення на основі хімічних реакцій, низькі температури можуть знизити їх ефективність і ємність. Перешкоди в роботі жорстких дисків, низькі температури можуть впливати на роботу жорстких дисків, викликаючи зміни в їх механічних властивостях та підвищуючи ризик виникнення помилок при запису або зчитуванні даних. Отже контроль температури в серверній кімнаті є критично важливим для забезпечення оптимальної роботи серверного обладнання.

Також важливим є спостереження за рівнем напруги та струму в мережі, до якої підключений сервер. Спостереження за напругою в серверній кімнаті є критично важливим з кількох причин, пов’язаних з забезпеченням безпеки, стабільності та надійності роботи серверного обладнання. До прикладу перенапруга, різке підвищення напруги може пошкодити чутливі електронні компоненти серверів. Це може призвести до виходу з ладу важливих частин, таких як материнські плати, процесори та жорсткі диски. Також можуть бути проблеми і через недонапругу, занадто низька напруга може спричинити нестабільну роботу серверів, призводячи до збоїв системи, помилок в роботі програмного забезпечення та втрати даних. Потрібно забезпечити стабільну роботу системи, сервери потребують стабільного джерела живлення для забезпечення безперебійної роботи. Коливання напруги можуть викликати перезавантаження або зупинку системи, що є критичним для серверів, які

обслуговують важливі бізнес-процеси. Також не можна забувати про захист від коротких замикань та електричних пожеж, неналежне спостереження за напругою може призвести до коротких замикань, що можуть спричинити пожежу в серверній кімнаті. Спостереження за напругою допомагає виявити та запобігти таким ситуаціям заздалегідь. Зажвжди є шляхи оптимізації енергоспоживання, спостереження за напругою допомагає виявити ефективність використання енергії серверним обладнанням. Це дозволяє оптимізувати енергоспоживання, що може призвести до зниження витрат на електроенергію та підвищення загальної ефективності роботи серверної інфраструктури. Ще дуже важливим є попередження збоїв у роботі мережі, нестабільність напруги може впливати на роботу мережевого обладнання, такого як маршрутизатори, комутатори та мережеві сховища даних. Це може призвести до збоїв у роботі мережі, втрати зв’язку та даних, тому потрібно подбати і про цей момент.

Звичайно потрібно подбати і про контроль вологості повітря в серверній кімнаті. Адже це є надзвичайно важливим для забезпечення продуктивної, безпечної та стабільної роботи серверів. Так як, висока вологість може спричиняти корозію металевих деталей серверного обладнання, що погано вплине на їх ефективність і скоротить термін експлуатації, тобто скоріш за все це призведе до виведення з ладу комплектуючої чи комплектуючих. Через надмірну вологість буде утворюватися конденсат на внутрішніх компонентах, що, теоретично, може спричинити коротке замикання, що за собою тягне серйозні пошкодження обладнання, а також втрату можливу втрату даних. Високий рівень вологості також порушує роботу систем охолодження, підвищуючи загальне енергоспоживання і знижуючи продуктивність систем охолодження, це в свою чергу може призводити до перегріву обладнання. Оптимальним рівнем вологості для серверного приміщення є 40-60%. Врахування і дотримання цих аспектів сприятиме стабільній і надійній роботі серверів, підвищуючи їх продуктивність і тривалість служби. Тому, контроль

вологості в серверному приміщенні є одним з найважливіших для підтримки надійної та ефективної роботи обладнання.

Дуже розповсюдженою проблемою серверних кімнат, при неналежному підході до безпеки, є пожежі. Пожежа може знищити все обладнення, а ще гірше, може нашкодити якійсь людині. Тож в нашому приладі є датчик вогню, який буде попереджувати такі випадки. За допомогою Blynk, власнику будуть надсилатися сповіщення про пожежу на телефон, що в свою чергу значно скоротить час реагування, і тоді можна буде врятувати не тільки обладнення а й чиєсь життя.

1.2 Середовище для розробки проекту

Для реалізації роботи цього пристрою можна використовувати багато різних середовищ розробки, такі як:

- PlatformIO [4];
- Espressif IoT Development Framework (ESP-IDF) [5];
- MicroPython або CircuitPython [6];

PlatformIO - це середовище розробки для вбудованих систем, яке підтримує ESP32 [7]. Воно інтегрується з різними редакторами коду, такими як Visual Studio Code, Atom та Sublime Text. PlatformIO дозволяє легко керувати бібліотеками та залежностями проекту, а також надає багато інструментів для роботи з ESP32.

Espressif IoT Development Framework (ESP-IDF) - це офіційне середовище розробки від Espressif для ESP32. ESP-IDF надає низькорівневий доступ до функцій ESP32 та дозволяє розробляти складні додатки, такі як мережеві проекти, з використанням різних протоколів та функціоналу пристрою ESP-IDF підтримує мову програмування C та C++.

MicroPython або CircuitPython - якщо вам подобається програмування мовою Python, ви можете використовувати MicroPython або CircuitPython на ESP32. MicroPython або CircuitPython дозволяє вам швидко створювати складні проекти на мікроконтролерах, таких як ESP32, використовуючи простий та зрозумілий синтаксис Python.

Проте було обрано - Arduino IDE, вона відома своєю простотою та зручністю для користувачів різного рівня [8]. Вона пропонує інтуїтивно зрозумілий інтерфейс. Завдяки простій структурі та наявності великої кількості прикладів і бібліотек є можливість швидко розрбляти складні проекти. Також Arduino IDE має велику і активну спільноту розробників. Це означає, що існує безліч форумів, туторіалів та прикладів коду, доступних онлайн. Така підтримка дозволяє швидко знаходити відповіді на питання та вирішувати проблеми, з якими можуть стикатися розробники під час роботи з ESP32. Це середовище розробки, спрощує інтеграцію різних апаратних компонентів через велику кількість готових бібліотек. Наприклад, бібліотеки для роботи з сенсорами, дисплеями, модулями зв’язку та іншими компонентами легко додаються та використовуються в проектах. Це дозволяє швидко розширювати функціонал пристрою без необхідності писати код з нуля. Не менш важливим є те, що Arduino IDE підтримує безліч бібліотек, які роблять процес розробки швидким і ефективним. Бібліотеки для ESP32 забезпечують легкий доступ до функціоналу мікроконтролера, такого як Wi-Fi, Bluetooth, аналогові та цифрові інтерфейси. Більшість бібліотек легко встановлюються через вбудований менеджер бібліотек, що значно спрощує процес розробки. Також важлою перевагою є те, що Arduino IDE працює на різних операційних системах, включаючи Windows, macOS та Linux. Це забезпечує гнучкість для розробників, які можуть працювати на будь-якій зручній платформі. Така кросплатформеність робить Arduino IDE універсальним інструментом для різних середовищ розробки.

1.3 Бібліотеки для роботи з додатковим обладанням

Після вибору середовища необхідно обрати бібліотеки, за допомогою яких буде розроблятись проект. Перша бібліотека яку обрано для проекта це LiquidCrystal_I2C (зображено на рис. 1.1) [9, 10]. Бібліотека LiquidCrystal_I2C використовується для роботи з LCD-дисплеями через інтерфейс I2C. Вона дозволяє легко підключати дисплей до мікроконтролера та керувати ним.

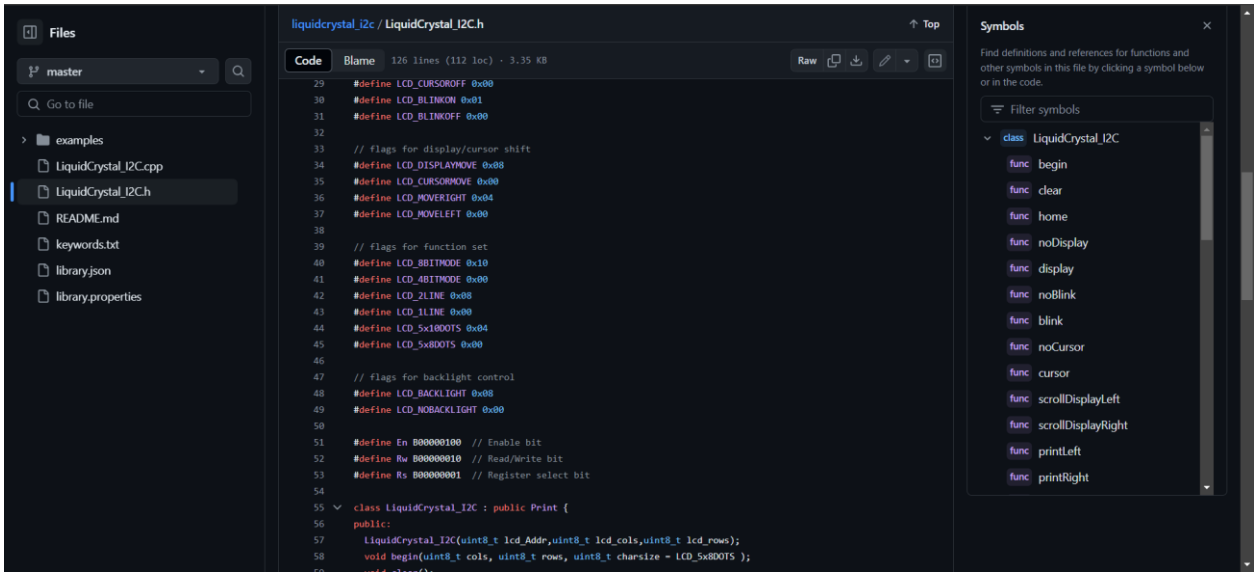


Рисунок 1.1 - LiquidCrystal_I2C, репозиторій GitHub, файл LiquidCrystal_I2C.h.

Adafruit_LiquidCrystal (зображено на рис. 1.2) - ця бібліотека також підтримує роботу з LCD-дисплеями через I2C та SPI [11, 12]. Вона надає більш розширені можливості для конфігурації та налаштування дисплеїв. З переваги, підтримка SPI, велика кількість додаткових функцій. Проте є й недоліки, вона більш складна у використанні, особливо для новачків.

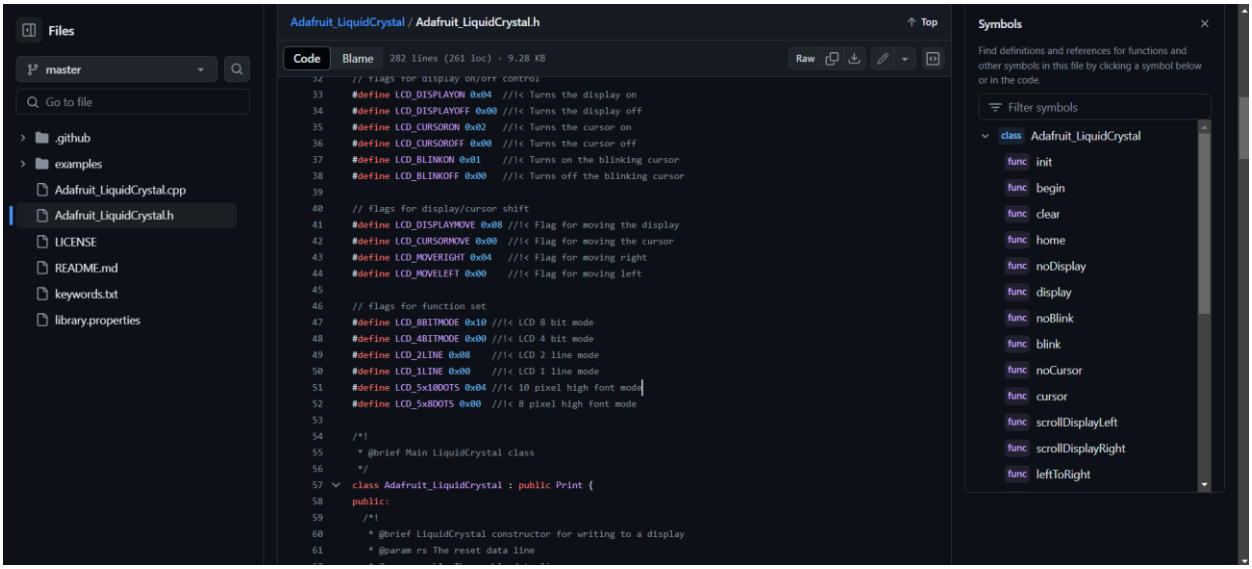


Рисунок 1.2 - Adafruit_LiquidCrystal, репозиторій GitHub, файл Adafruit_LiquidCrystal.h.

Є ще один аналог – NewLiquidCrystal (зображено на рис. 1.3) [13]. Це бібліотека, яка є розширеною версією стандартної LiquidCrystal і підтримує I2C, SPI та паралельний інтерфейс. З переваг, у неї більша гнучкість у виборі інтерфейсів, активна підтримка. Але також є недоліки, може вимагати більше налаштувань для роботи.

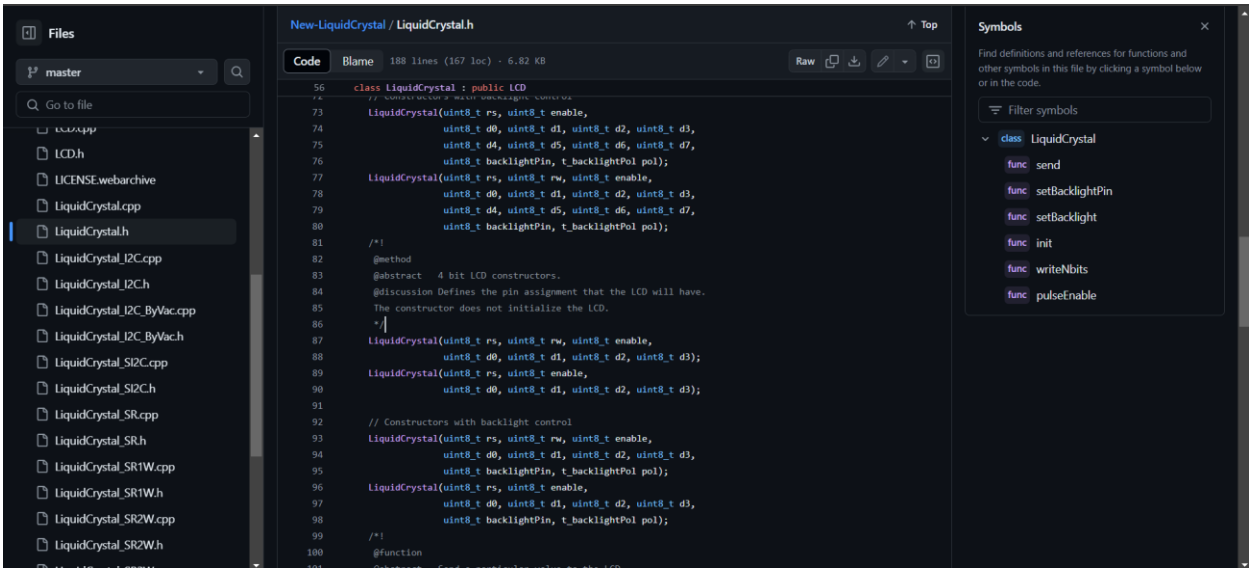


Рисунок 1.3 - NewLiquidCrystal, репозиторій GitHub, LiquidCrystal.h.

Головною причиною вибору LiquidCrystal_I2C є простота використання. Бібліотека LiquidCrystal_I2C є дуже легкою у використанні та налаштуванні, що дозволяє швидко й без зайвих проблем інтегрувати LCD-дисплей у проект. У неї більш ніж достатні можливості. Вона надає всі необхідні нам функції для роботи з LCD через I2C, що підходить для більшості стандартних завдань, в тому числі для нашого.

Наступна бібліотека – DHT. Бібліотека DHT використовується для зчитування даних з датчиків температури і вологості серії DHT, таких як DHT11 і DHT22 (зображено на рис. 1.4) [14, 15]. Ці датчики широко використовуються в різних проектах з мікроконтролерами завдяки їх простоті та доступності. Бібліотека забезпечує простий інтерфейс для взаємодії з цими датчиками, дозволяючи легко отримувати дані про температуру та вологість. Датчики серії DHT (Digital Humidity and Temperature) є цифровими датчиками, що використовуються для вимірювання температури та вологості. Найпоширенішими моделями є DHT11 і DHT22. У DHT11 діапазон температури: 0-50°C з точністю ±2°C та діапазон вологості: 20-80% з точністю ±5%. У цієї моделі є певні переваги, вона дешевша та простіша у використанні. Але є і недоліки, в неї менша точність та діапазон вимірювань порівняно з DHT22. У DHT22 (або AM2302) діапазон температури: -40-80°C з точністю ±0.5°C та діапазон вологості: 0-100% з точністю ±2-5%. З переваг, цей датчик більш точний і в нього широкий діапазон вимірювань. Проте він дорожчий та трохи складніший у використанні.

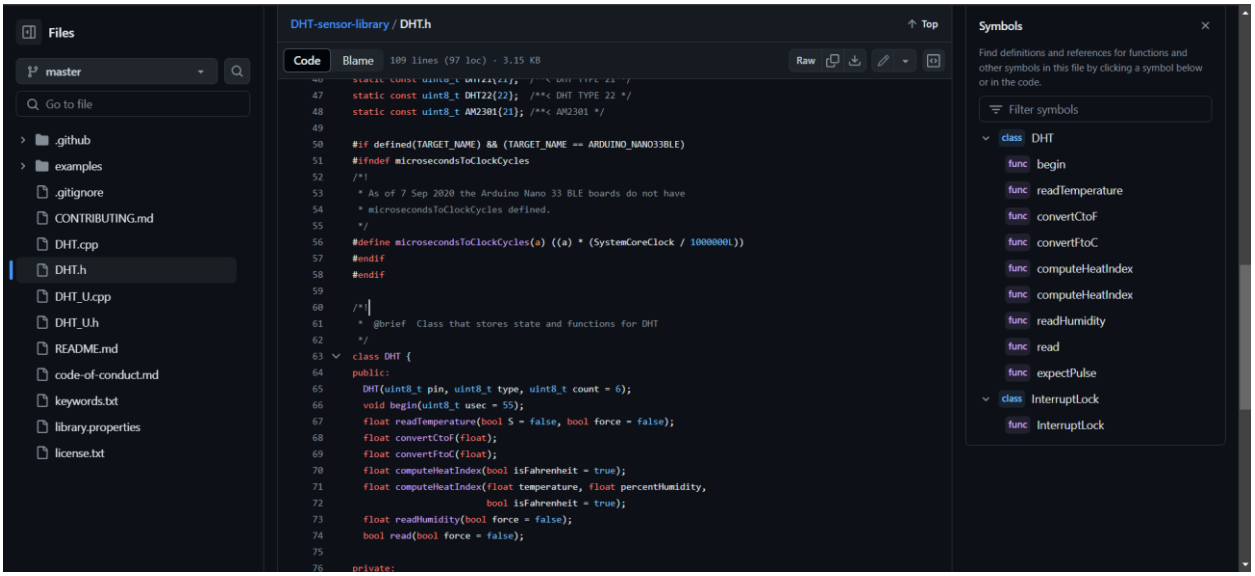


Рисунок 1.4 - DHT, репозиторій GitHub, файл DHT.h.

У бібліотеки DHT також є безліч аналогів. Adafruit_Sensor (зображено на рис. 1.5) - це бібліотека для роботи з різними датчиками від Adafruit, включаючи серію DHT [16, 17]. Вона надає уніфікований інтерфейс для роботи з різними сенсорами. Вона підтримує багато різних датчиків, що дозволяє легко інтегрувати різні сенсори в один проект. Також у неї відмінна документація і приклади від Adafruit, що робить її зручною для використання. Проте вона може бути складнішою для новачків через загальний підхід до роботи з датчиками. Також потрібно буде використовувати додаткову бібліотеку (Adafruit Unified Sensor), що збільшує розмір програми.

Є ще один аналог – SimpleDHT (зображено на рис. 1.6) [18, 19]. Легка бібліотека, яка зосереджується на простоті та швидкості взаємодії з датчиками DHT. Її основна перевага у простоті використанні, та мінімальні залежності. Легкий код, який робить її ідеальною для проектів з обмеженими ресурсами. Але вона менш потужна у порівнянні з більш розширеними бібліотеками. Також у неї досить обмежені функції і можливості порівняно з іншими бібліотеками.

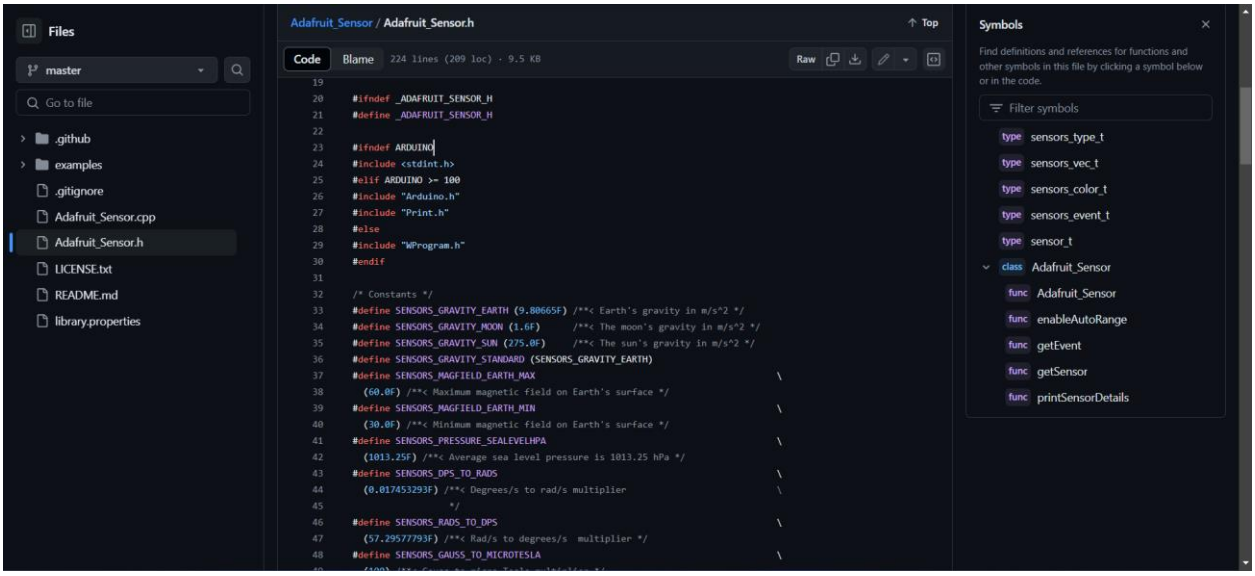


Рисунок 1.5 - Adafruit_Sensor, репозиторій GitHub, файл Adafruit_Sensor.h .

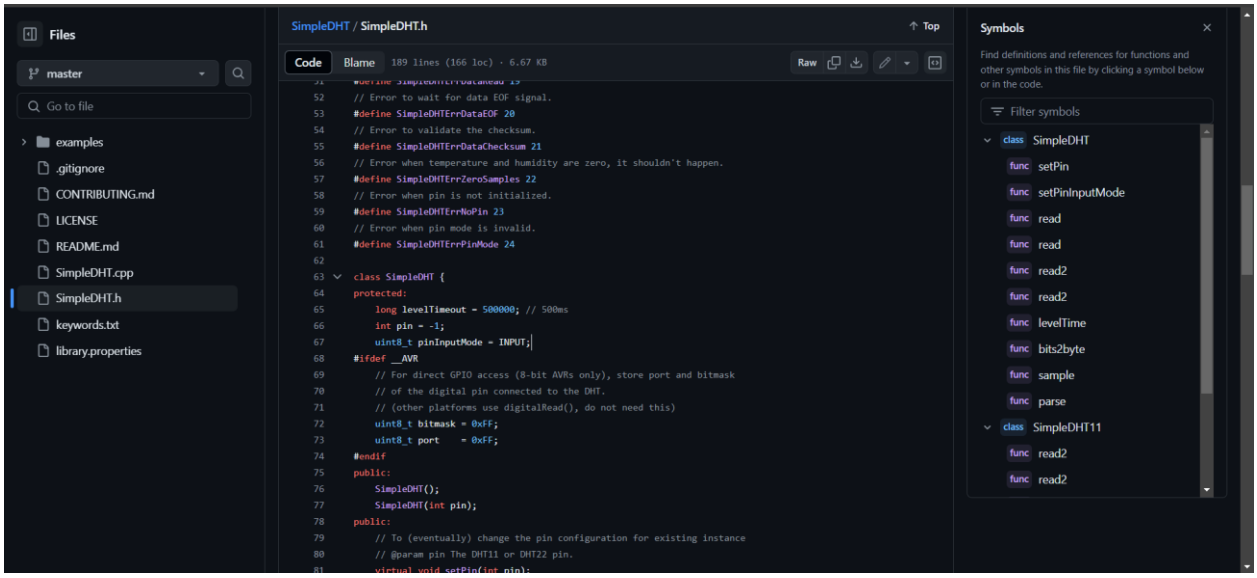


Рисунок 1.6 – SimpleDHT, репозиторій GitHub, файл SimpleDHT.h.

Бібліотека DHT проста та ефективна, бібліотека DHT забезпечує простий інтерфейс для зчитування даних з датчиків DHT11 і DHT22. Вона легка у використанні і не потребує додаткових налаштувань. Також не мало важливий факт, ця бібліотека широко використовується в спільноті розробників Arduino, що забезпечує велику кількість прикладів та підтримку. Це важливо для швидкого вирішення проблем та інтеграції в проекти. Бібліотека DHT є дуже

стабільною і добре протестованою, що робить її неймовірно надійним вибором для вимірювання температури та вологості у різних проектах. Бібліотека DHT є важливим інструментом для роботи з датчиками серії DHT, забезпечуючи простий і ефективний спосіб отримання даних про температуру та вологість. Завдяки своїй простоті, стабільності та широкій підтримці, вона є популярним вибором серед розробників, особливо тих, хто працює з проектами на базі мікроконтролерів Arduino та ESP32.

Наступною бібліотекою є WiFi.h (зображено на рис. 1.7) [20, 21]. Бібліотека WiFi.h є стандартною бібліотекою для роботи з Wi-Fi на мікроконтролерах ESP32. Вона забезпечує інтерфейси для підключення до Wi-Fi мереж, управління підключеннями, отримання інформації про мережі та інші функції, пов'язані з Wi-Fi. Однією з переваг цієї бібліотеки є повна функціональність, бібліотека підтримує всі необхідні функції для роботи з Wi-Fi, включаючи режим клієнта та точки доступу. Також її досить просто використовувати завдяки інтуїтивно зрозумілому синтаксису, що спрощує розробку мережевих програм. WiFi.h спеціально розроблена для роботи з ESP32, що забезпечує високу сумісність та стабільність. Проте у неї є і недоліки. Ця бібліотека може займати значну кількість пам'яті та процесорного часу, що може бути критичним для обмежених ресурсів мікроконтролера. Досить обмежена документація, хоча основні функції добре документовані, інколи виникає необхідність у додаткових джерелах для розуміння більш складних аспектів.

Є ще один аналог WiFi.h – це ESP8266WiFi.h (зображено на рис. 1.8) [22, 23]. Хоча ESP8266WiFi.h в основному використовується для платформи ESP8266, вона також може працювати з ESP32 за допомогою відповідних адаптацій. Ця бібліотека забезпечує схожі функції для роботи з Wi-Fi мережами. Бібліотека має дуже схожий інтерфейс до WiFi.h, що спрощує перехід між платформами. Також у цієї бібліотеки, хороша документація, існує багато ресурсів та прикладів для цієї бібліотеки. З недоліків є те, що менша підтримка

ESP32, бібліотека в основному орієнтована на ESP8266, що може призводити до обмежень при використанні на ESP32.

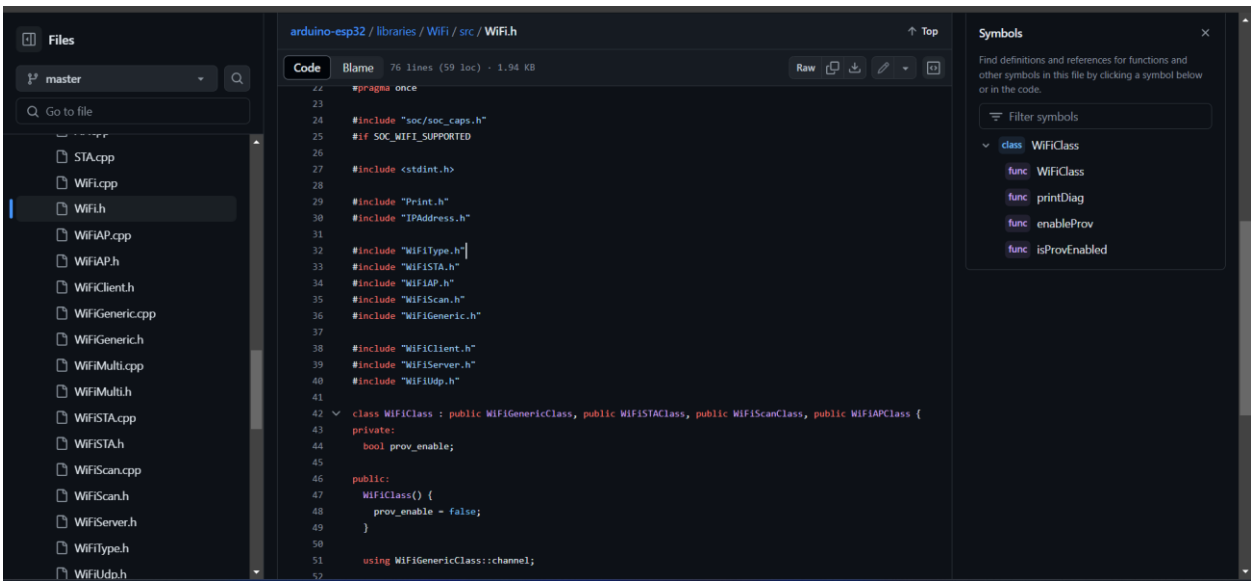


Рисунок 1.7 – WiFi.h, репозиторій GitHub, файл WiFi.h.

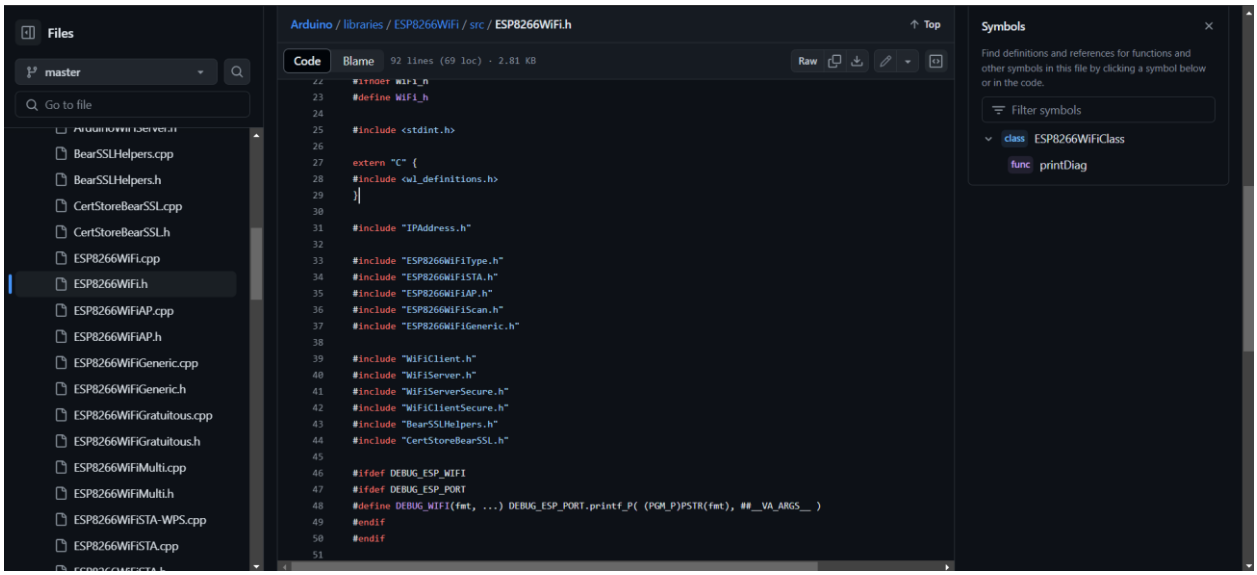


Рисунок 1.8 – ESP8266WiFi.h, репозиторій GitHub, файл ESP8266WiFi.h.

Вибір бібліотеки WiFi.h для нашого проекту був зумовлений кількома ключовими факторами. Це сумісність з ESP32, ця бібліотека спеціально розроблена для роботи з ESP32, що забезпечує високу стабільність та сумісність.

Також WiFi.h надає всі необхідні функції для роботи з Wi-Fi мережами, що дозволяє реалізувати всі необхідні аспекти проекту без додаткових бібліотек. Також ця бібліотека є простою у використанні, її інтуїтивно зрозумілий синтаксис та велика кількість доступних прикладів дозволяють швидко розробити та налаштувати мережеві функції вашого пристрою. Таким чином, вибір бібліотеки WiFi.h є обґрунтованим та логічним рішенням для забезпечення надійного та ефективного підключення нашого ESP32 до Wi-Fi мережі.

Бібліотека WiFiClient.h (зображено на рис. 1.9) [24, 25]. Вона є частиною стандартної бібліотеки Wi-Fi для ESP32. Вона надає функції для створення клієнтських з'єднань через Wi-Fi. Використовується для встановлення TCP-з'єднань з серверами, що дозволяє передавати та отримувати дані по мережі. Ця бібліотека, дуже легко інтегруєтьсяЛегка з іншими бібліотеками Wi-Fi, такими як WiFi.h, що дає нам можливість забезпечити повну мережеву функціональність. Також вона широко використовується спільнотою розробників, що забезпечує велику кількість прикладів та документації. Не менш важливим фактором є те, що ця бібліотека підтримує основні мережеві протоколи, необхідні для роботи з різними серверами та сервісами.

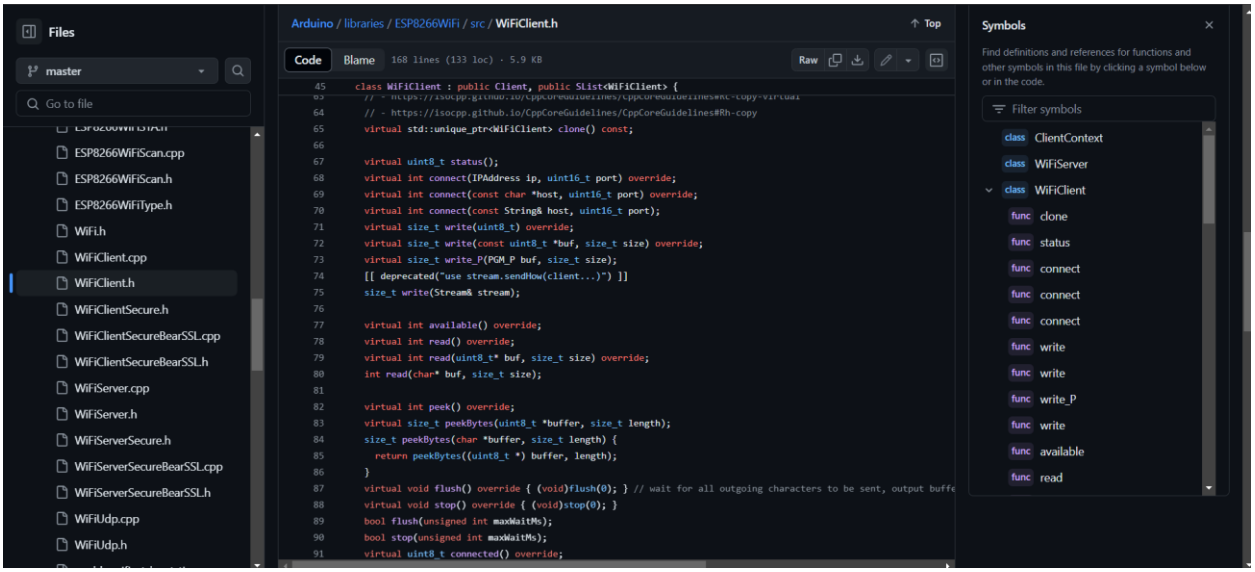


Рисунок 1.9 – WiFiClient.h, репозиторій GitHub, файл WiFiClient.h.

Також є аналог - це бібліотека AsyncTCP.h (зображено на рис. 1.10), вона є асинхронною бібліотекою для роботи з TCP-з'єднаннями, що надає можливість асинхронної обробки даних [26, 27]. Асинхронна обробка, дозволяє обробляти дані асинхронно, що могло б покращити продуктивність нашого додатку. Менше навантаження на процесор під час обробки великих обсягів даних, це також добре в плинуло б на продуктивність. Проте вона досить складна у використанні. Асинхронний підхід складніший для розуміння та реалізації, особливо для початківців. Також не дивлячись на крутість асинхронних операцій, їх використання може призводити до додаткового використання пам'яті та процесорного часу, що може вплинути на ресурсоємність.

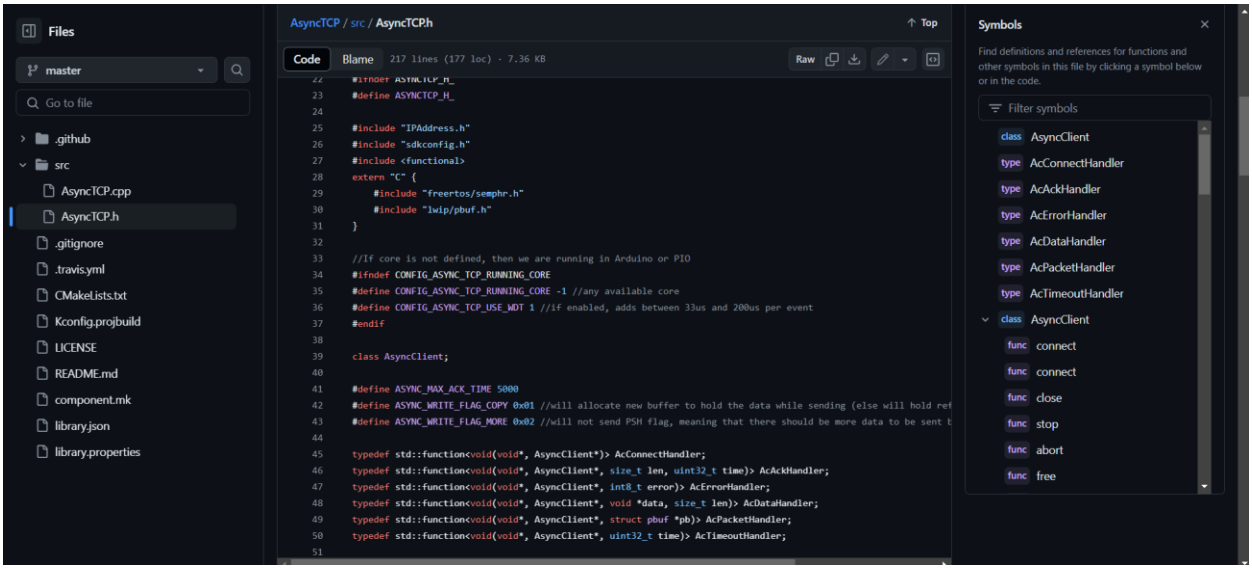


Рисунок 1.10 – AsyncTCP.h, репозиторій GitHub, файл AsyncTCP.h.

Проте було обрано бібліотеку WiFiClient.h для нашого проекту. WiFiClient.h легко інтегрується з іншими стандартними бібліотеками ESP32, такими як WiFi.h, що забезпечує зручність використання та стабільність. Велика кількість прикладів та документації дозволяє швидко знайти рішення для виникаючих проблем, що знижує час на розробку та налагодження. Також WiFiClient.h надає всі необхідні функції для реалізації клієнтських TCP-з'єднань, що відповідає вимогам нашого проекту. Отже, використання бібліотеки

WiFiClient.h у нашому проекті є оптимальним вибором, оскільки вона забезпечує необхідну функціональність, стабільність та простоту інтеграції, що робить її зручним інструментом для розробки мережевих додатків на платформі ESP32.

Наступною бібліотекою, яка є необхідною для розробки є HardwareSerial.h (зображено на рис. 1.11) [28, 29]. Бібліотека HardwareSerial.h є стандартною для роботи з апаратним послідовним інтерфейсом (UART) на платформах Arduino, включаючи ESP32. Вона дозволяє створювати з'єднання через послідовний порт для обміну даними з іншими пристроями, такими як модулі датчиків, комп'ютери або інші мікроконтролери. Вона проста у використанні, має доступний і зрозумілий інтерфейс для налаштування та використання послідовного порту. Підтримується на більшості платформ Arduino, включаючи ESP32. Дозволяє низькорівневий доступ до апаратного UART, забезпечуючи високу гнучкість у налаштуванні. Також використання апаратного UART забезпечує стабільну та надійну передачу даних без зайвих затримок.

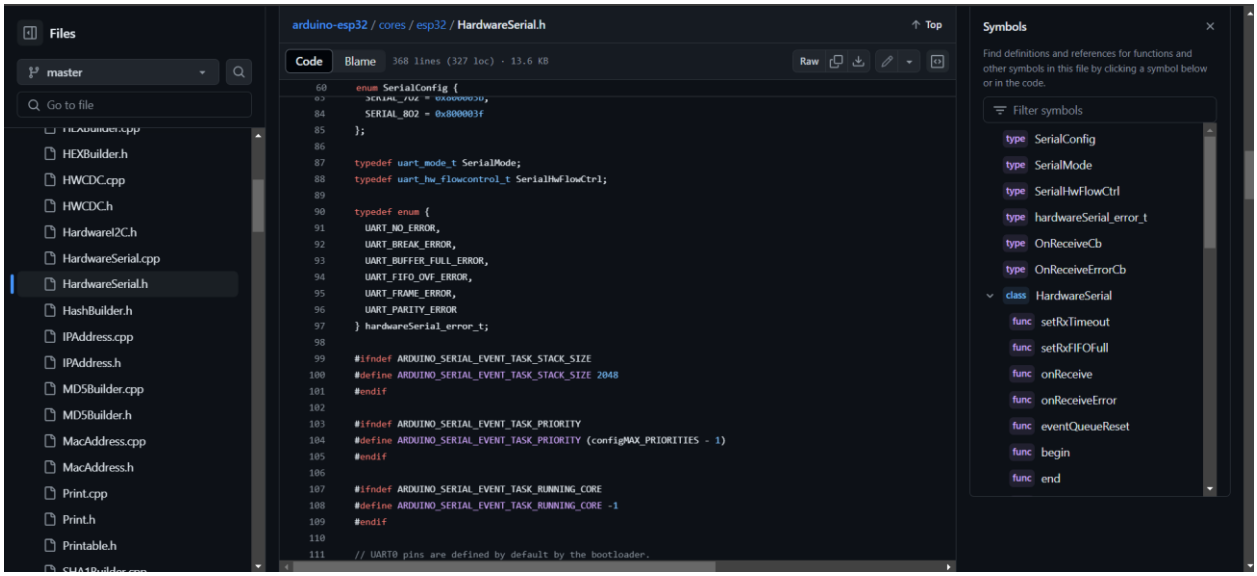


Рисунок 1.11 – HardwareSerial.h, репозиторій GitHub, файл HardwareSerial.h.

Головним аналогом є SoftwareSerial.h (зображено на рис. 1.12, 1.13) [30, 31]. Це бібліотека для створення додаткових послідовних портів через програмні

засоби, що дозволяє використовувати більше UART з'єднань. Вона дозволяє створювати декілька послідовних портів, використовуючи програмні засоби. Надає можливість використання будь-яких цифрових пінів для послідовної комунікації. Проте у неї низька швидкість. Вона менш ефективна у порівнянні з апаратним UART, особливо на високих швидкостях передачі даних. Ще однією проблемою є те, що вона використовує процесорні ресурси для емуляції UART, що може знижувати продуктивність.

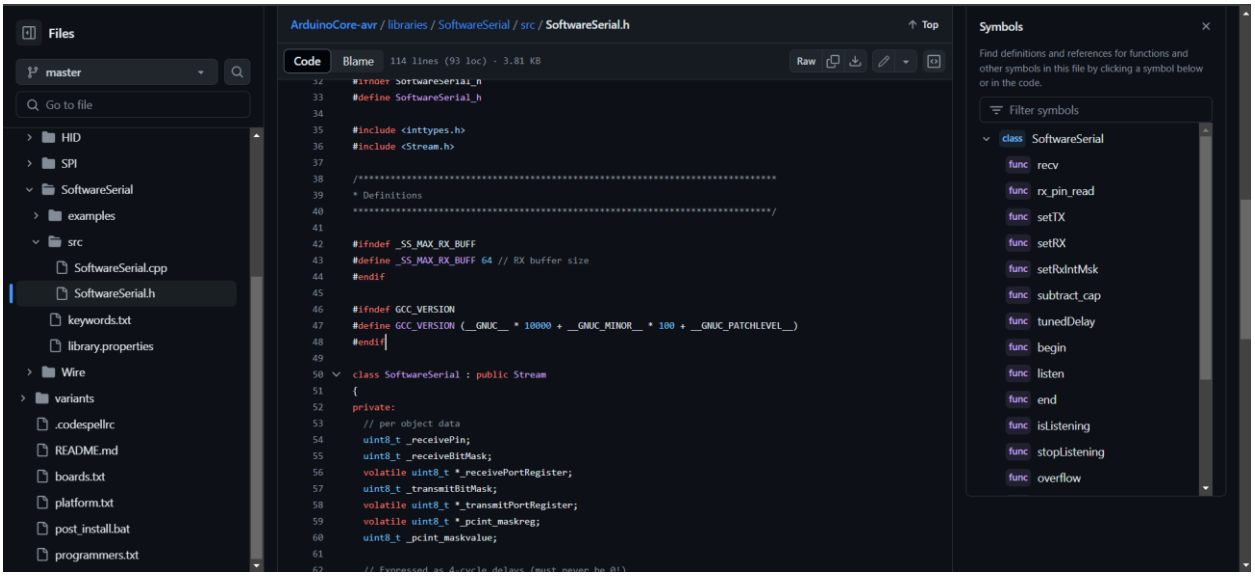


Рисунок 1.12 – SoftwareSerial.h, репозиторій GitHub, файл SoftwareSerial.h.

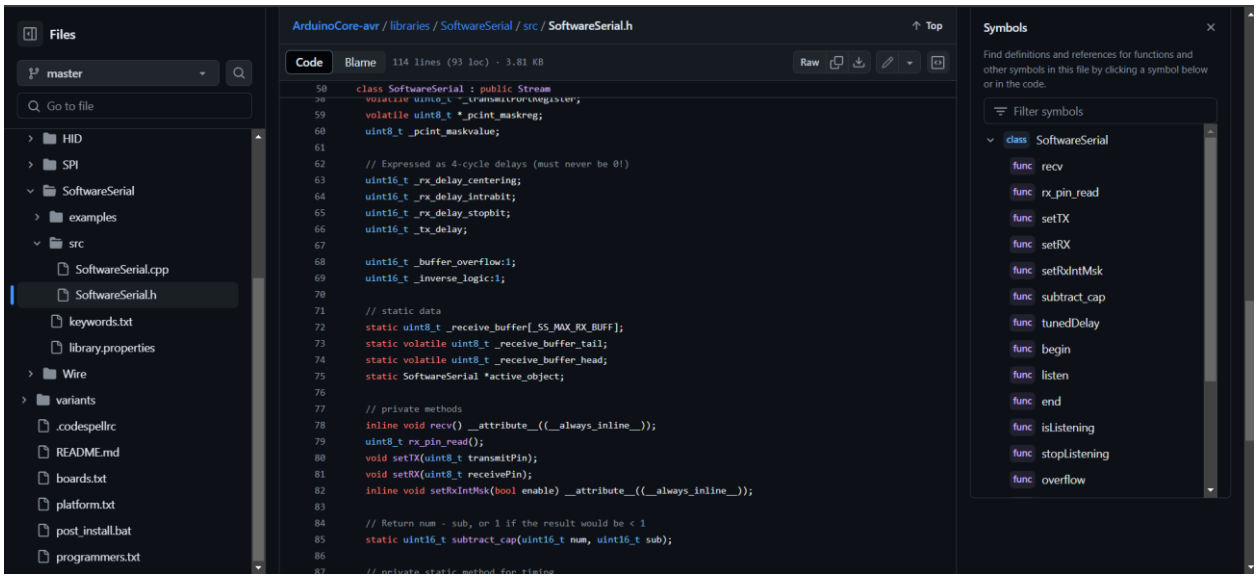


Рисунок 1.13 – SoftwareSerial.h, репозиторій GitHub, файл SoftwareSerial.h.

Зваживши все, було обрано бібліотеку HardwareSerial.h. Просто та стабільна, HardwareSerial.h забезпечує доступний та стабільний інтерфейс для роботи з апаратним UART, що робить її зручною для використання в проектах будь-якої складності. Бібліотека підтримується на більшості платформ Arduino, включаючи ESP32, що гарантує сумісність та надійність. Також вона надає необхідну функціональність для нашого проекту. Нам не потрібні асинхронні операції або додаткові послідовні порти, тому HardwareSerial.h є оптимальним вибором, забезпечуючи всі необхідні функції. Ще одним важливим плюсом є те, що Бібліотека HardwareSerial.h легко інтегрується з іншими компонентами нашого проекту, такими як PZEM004Tv30 для моніторингу електричних параметрів, що підвищує загальну ефективність нашої системи. Бібліотека HardwareSerial.h є оптимальним вибором для вашого проекту через її простоту використання, надійність та інтеграцію з апаратним UART ESP32. Хоча є альтернативи, такі як SoftwareSerial.h та інші, для нашого конкретного завдання бібліотека HardwareSerial.h забезпечує необхідну функціональність та стабільність без додаткових складнощів.

Ще залишилося обрати бібліотеку для вимірювання напруги, струму ітд. Мій вибір впав на PZEM004Tv30.h (зображено на рис. 1.14), вона використовується для взаємодії з модулем PZEM-004T v3.0, який є вимірювачем електричних параметрів, таких як напруга, струм, потужність та енергія [32, 33]. Цей модуль зазвичай використовується для моніторингу електричних мереж у реальному часі. Доступна, легка інтеграція з ESP32 через послідовний інтерфейс (UART), що робить її зручною для використання у проектах Arduino. Бібліотека забезпечує функції для зчитування всіх ключових параметрів електричної мережі. Також висока надійність і стабільність у передачі даних між модулем і мікроконтролером. Також у неї дуже детальна документація. Бібліотека добре документована, що спрощує її використання та налаштування.

```
    float power = pzem.power();
    float energy = pzem.energy();
    float frequency = pzem.frequency();
    float pf = pzem.pf();

    // Check if the data is valid
    if(isnan(voltage)){
        Serial.println("Error reading voltage");
    } else if (isnan(current)) {
        Serial.println("Error reading current");
    } else if (isnan(power)) {
        Serial.println("Error reading power");
    } else if (isnan(energy)) {
        Serial.println("Error reading energy");
    } else if (isnan(frequency)) {
        Serial.println("Error reading frequency");
    } else if (isnan(pf)) {
        Serial.println("Error reading power factor");
    } else {

        // Print the values to the Serial console
        Serial.print("Voltage: ");      Serial.print(voltage);      Serial.println("V");
        Serial.print("Current: ");      Serial.print(current);      Serial.println("A");
        Serial.print("Power: ");        Serial.print(power);        Serial.println("W");
        Serial.print("Energy: ");       Serial.print(energy,3);     Serial.println("kWh");
        Serial.print("Frequency: ");    Serial.print(frequency, 1); Serial.println("Hz");
        Serial.print("PF: ");           Serial.println(pf);

    }

    Serial.println();
    delay(2000);
```

Рисунок 1.14 – PZEM004Tv30.h, репозиторій GitHub, файл PZEM004Tv30.h.

З аналогів є EmonLib (зображено на рис. 1.15) [34, 35]. Бібліотека EmonLib є популярною альтернативою для моніторингу електричних параметрів. Вона розроблена для використання з різними типами датчиків напруги та струму і часто використовується в проектах з відкритим кодом для енергомоніторингу. Вона досить гнучка, підтримує широкий спектр датчиків, включаючи трансформатори струму та датчики напруги. Також у неї відкрите програмне забезпечення. EmonLib є частиною проекту OpenEnergyMonitor, що забезпечує доступ до великої кількості ресурсів та спільноти. Також добре масштабується, тож підходить для моніторингу складних систем з кількома датчиками. Однак вона є досить складною в налаштуваннях, вимагає додаткових налаштувань та калібрування для роботи з різними датчиками. Досить обмежена точність, вона може мати нижчу точність порівняно з спеціалізованими модулями, такими як

PZEM-004T v3.0. Ще потребує використання зовнішніх датчиків та схеми для правильного вимірювання, що ускладнює реалізацію.

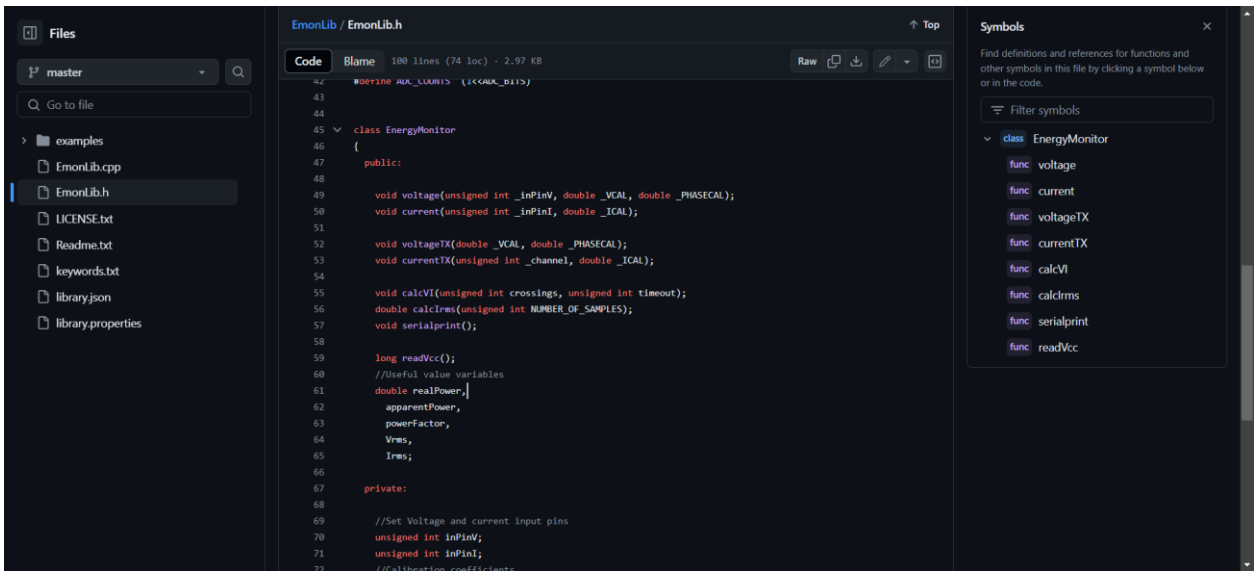


Рисунок 1.15 – EmonLib.h, репозиторій GitHub, файл EmonLib.h.

Бібліотека PZEM004Tv30.h була обрана для нашого проекту з кількох причин. Вона легко інтегрується з ESP32 через послідовний інтерфейс, що спрощує підключення та налаштування. Також використання модуля PZEM-004T v3.0 забезпечує високу точність та стабільність вимірювань, що є критично важливим для вашого проекту. Ще один вагомий плюс, це те, що ця бібліотека має добре задокументований інтерфейс і приклади використання, що зменшує час на її освоєння та налаштування. PZEM004Tv30.h забезпечує всі необхідні функції для моніторингу електричних параметрів, що робить її оптимальним вибором для нашого проекту без необхідності додаткових налаштувань та калібрувань. Отже бібліотека PZEM004Tv30.h є оптимальним вибором для нашого проекту через її простоту використання, високу точність та стабільність, що забезпечує модуль PZEM-004T v3.0. Незважаючи на наявність гнучкої альтернативи, такої як EmonLib, PZEM004Tv30.h надає всі необхідні функції та значно спрощує процес розробки та інтеграції в порівнянні з іншими рішеннями.

Залишилося проаналізувати останню бібліотеку, BlynkSimpleEsp32.h (зображено на рис. 1.16, 1.17) [36]. Бібліотека BlynkSimpleEsp32.h використовується для інтеграції платформи Blynk з мікроконтролером ESP32. Blynk — це платформа для інтернету речей (IoT), яка дозволяє керувати апаратними пристроями через інтернет, використовуючи мобільний додаток [37]. Вона забезпечує простий спосіб створення інтерфейсів для віддаленого керування і моніторингу. Blynk пропонує простий та доступний інтерфейс, детальну документацію, що робить процес налаштування легким навіть для людей, які вперше стикаються з таким завданням. Однією з переваг Blynk, є те, що він надає мобільні додатки для Android та iOS, які дозволяють швидко створювати користувацькі інтерфейси без необхідності програмування. Платформа має безліч віджетів, які можна використовувати для відображення даних, керування пристроями та взаємодії з користувачем. Також Blynk Cloud забезпечує надійний та швидкий зв'язок між мікроконтролером та мобільним додатком, зменшуючи необхідність у власному серверному рішенні. Blynk є досить гнучким, він підтримує різні платформи та мікроконтролери, включаючи ESP32, що робить його універсальним інструментом для IoT проектів. Також Blynk має велику спільноту користувачів і активну підтримку, що забезпечує швидке вирішення проблем і надання допомоги. Проте безкоштовна версія Blynk має обмежений функціонал і кількість віджетів, що може бути недостатнім для складних проектів, хоча в нашому випадку, ми вклалися в рамки безкоштовного користування. Також для використання повного функціоналу необхідне постійне підключення до інтернету і доступ до Blynk Cloud, що може бути проблематично в умовах ненадійного інтернет-з'єднання. Все ж таки, якщо б ми вирішили використовувати розширені функції і можливості Blynk, то нам потрібно було б купляти платну підписку, що може бути недоцільним для невеликих або навчальних проектів. Ще одним теоретичним недоліком може бути те, що використання Blynk означає

залежність від сторонньої платформи, що може стати проблемою в разі зміни політики компанії або припинення підтримки.

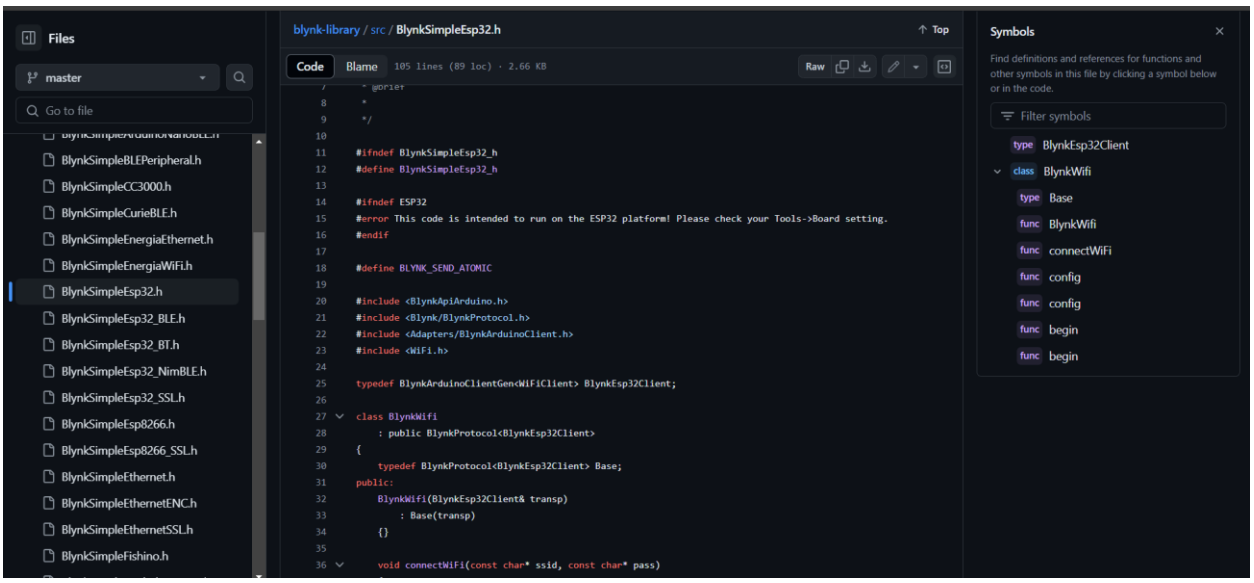


Рисунок 1.16 – BlynkSimpleEsp32.h, репозиторій GitHub, файл BlynkSimpleEsp32.h.

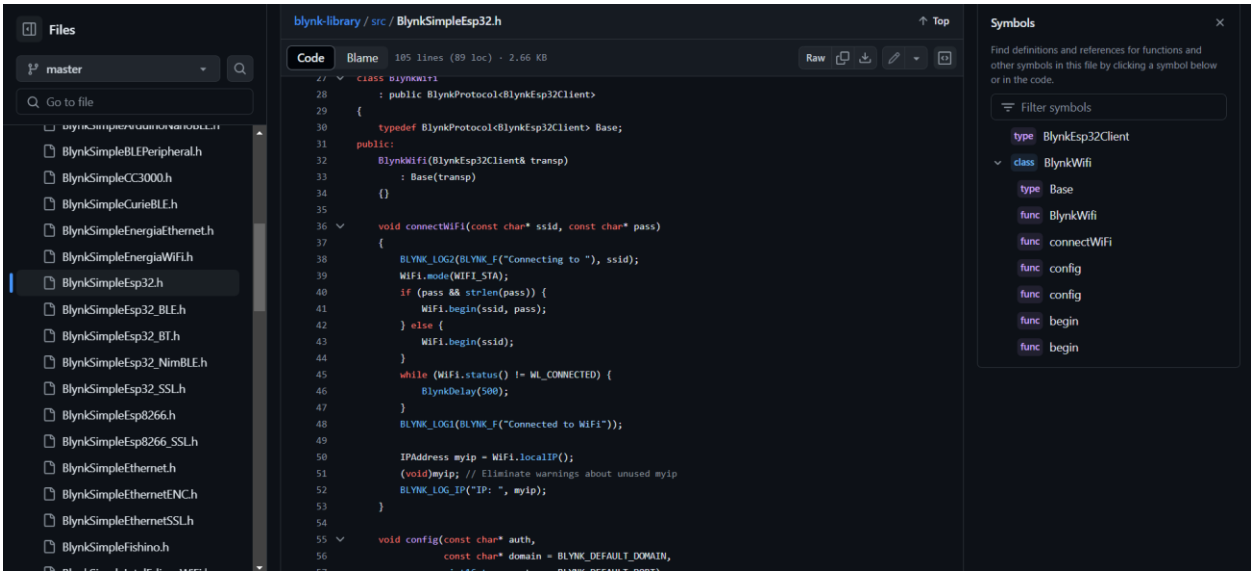


Рисунок 1.17 – BlynkSimpleEsp32.h, репозиторій GitHub, файл BlynkSimpleEsp32.h.

Натомість був аналог Firebase ESP32 (зображено на рис. 1.18) [38]. Бібліотека Firebase ESP32 дозволяє інтегрувати ESP32 з платформою Firebase.

Firebase — це платформа, створена Google, яка надає безліч інструментів для розробки, зберігання і аналізу даних для мобільних і веб-додатків [39]. Однією з основних переваг є розширені можливості зберігання даних. Firebase надає потужні можливості для зберігання і обробки даних в реальному часі. Досить логічним є те, що у Firebase легка інтеграція з іншими продуктами Google, такими як Google Analytics, Google Cloud та інші [40, 41]. Також Firebase забезпечує різноманітні методи автентифікації користувачів, такі як електронна пошта, Google Sign-In, Facebook та інші. Ще однім великим плюсом Firebase є те, що він надає потужні інструменти для аналітики і моніторингу додатків, проте в нашому випадку це не потрібно. Тим не менш налаштування Firebase складніше порівняно з Blynk. Ще одним недоліком є те, шо на відміну від Blynk, для Firebase необхідно розробляти власний мобільний додаток або веб-інтерфейс для взаємодії з користувачем. Використання Firebase означає тісну інтеграцію з іншими сервісами Google, що може бути недоцільним для деяких проектів, в тому числі і для нашого. Ну і звичайно вартість, деякі функції Firebase можуть вимагати платної підписки, що може бути проблемою для невеликих або навчальних проектів.

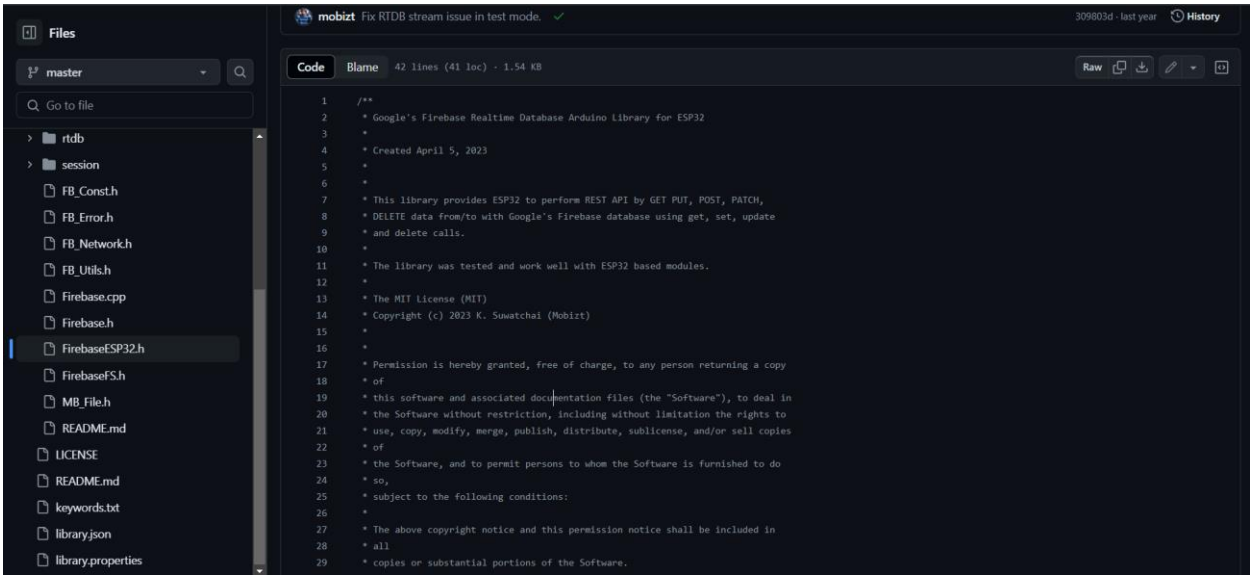


Рисунок 1.18 – Firebase ESP32.h, репозиторій GitHub, файл Firebase ESP32.h.

Blynk дозволяє швидко і легко інтегрувати IoT функціонал з мінімальними зусиллями. Це особливо важливо для проектів, де потрібно швидко створити прототип. Також він дуже зручний у використанні, завдяки мобільному додатку Blynk, користувач може легко керувати пристроєм без необхідності розробляти свій власний додаток. Широкий функціонал для IoT, Blynk пропонує великий набір віджетів для створення інтерфейсів, що дозволяє легко налаштовувати керування і моніторинг пристрою. Ну і останнє проте не найменш важливе, що Blynk має велику спільноту користувачів, що забезпечує швидке вирішення проблем та надання допомоги. Таким чином, для нашого проекту ми обрали бібліотеку BlynkSimpleEsp32.h через її простоту, зручність використання і спеціалізацію на IoT. Вона дозволяє швидко реалізувати необхідний функціонал з мінімальними зусиллями, що є важливим для ефективного і швидкого виконання нашого проекту.

2 РОЗРОБКА ПРОГРАМНОЇ СКЛАДОВОЇ СИСТЕМИ

2.1 Підключення бібліотек

Після вибору необхідних бібліотек, необхідно переходити до написання коду/логіки нашого приладу. Для розуміння роботи та функціоналу коду необхідно почати з опису підключених бібліотек. Кожна бібліотека забезпечує певний набір функцій та інтерфейсів для роботи з апаратними компонентами або програмними сервісами. Для того, щоб працювали екранчик, датчики, WiFi, порти, платформа Blynk, за допомогою, якої ми надсилаємо сповіщення, потрібні бібліотеки. Адже налаштовуваання цієї роботи вручну, забере дуже багато часу, тож краще використовувати готові і перевірені рішення.

Ось список біліотек, які ми будемо використовувати для написання логіки коду:

```
#include <LiquidCrystal_I2C.h>
#include "DHT.h"
#include <WiFi.h>
#include <WiFiClient.h>
#include <HardwareSerial.h>
#include <PZEM004Tv30.h>
#include <BlynkSimpleEsp32.h>
```

Розглянемо кожну бібліотеку.

```
#include <LiquidCrystal_I2C.h>
```

Бібліотека LiquidCrystal_I2C використовується для роботи з рідкокристалічними дисплеями (LCD), які підтримують інтерфейс I2C. Ця бібліотека значно спрощує роботу з дисплеями, дозволяючи розробникам легко виводити текстову інформацію на екран. Основні функції цієї бібліотеки включають ініціалізацію дисплея, встановлення курсору, очищення екрана та вивід тексту на дисплей.

Основні функції:

- `lcd.init()`;
- `lcd.backlight()`;
- `lcd.setCursor(column, row)`;
- `lcd.print("text")`;

lcd.init() - ця функція відповідає за ініціалізацію дисплея. lcd.backlight() - ця функція відповідає за включення підсвічування дисплея. lcd.setCursor(column, row) - відповідає за встановлення курсору в позицію (column, row). lcd.print("text") - відповідає за виведення тексту на дисплей.

Ця бібліотека використовується для виведення на дисплей таких параметрів, як температура, вологість, напруга та струм, що дозволяє користувачу зручно бачити поточні вимірювання безпосередньо на самому пристрої.

```
#include "DHT.h"
```

Бібліотека DHT призначена для роботи з датчиками температури та вологості серії DHT, таких як DHT11, DHT22 тощо. Вона надає функції для зчитування температури та вологості з цих датчиків.

Основні функції:

- `dht.begin()`;
- `dht.readTemperature()`;
- `dht.readHumidity()`;

dht.begin() - ця функція відповідає за ініціалізацію датчика. dht.readTemperature() - відповідає за зчитування температури. dht.readHumidity() - відповідає за зчитування вологості.

Ця бібліотека використовується для отримання даних про температуру та вологість, що є критично важливими параметрами для моніторингу стану приміщення з серверним обладнанням.

```
#include <WiFi.h>
```

Бібліотека WiFi використовується для забезпечення бездротового підключення до мережі Wi-Fi. Вона надає функції для підключення до Wi-Fi мережі, управління з'єднанням та обміну даними через Інтернет.

Основні функції:

- WiFi.begin(ssid, password);
- WiFi.status();
- WiFi.disconnect();

WiFi.begin(ssid, password) - ця функція відповідає за підключення до Wi-Fi мережі з вказаними SSID та паролем. WiFi.status() - ця функція відповідає за отримання статусу з'єднання. WiFi.disconnect() - ця функція відповідає за Відключення від Wi-Fi мережі.

Ця бібліотека необхідна для забезпечення підключення пристрою до Інтернету через Wi-Fi, що дозволяє передавати дані на сервер Blynk для віддаленого моніторингу та керування.

```
#include <WiFiClient.h>
```

Бібліотека WiFiClient є частиною бібліотеки WiFi і надає клас для створення клієнтських з'єднань з сервером через Wi-Fi. Вона використовується для встановлення та управління TCP-з'єднаннями.

Основні функції:

- WiFiClient.connect(host, port);
- WiFiClient.write(data);
- WiFiClient.read();

WiFiClient.connect(host, port) - відповідає за підключення до сервера з вказаним хостом та портом. WiFiClient.write(data) - відповідає за відправку даних на сервер. WiFiClient.read() - відповідє за зчитування даних із сервера.

Ця бібліотека забезпечує передачу даних на сервер Blynk, що дозволяє відправляти виміряні параметри та отримувати команди для керування пристроєм.

```
#include <HardwareSerial.h>
```

Бібліотека HardwareSerial надає інтерфейс для роботи з апаратними серійними портами мікроконтролера. Вона дозволяє налаштовувати та використовувати серійні порти для обміну даними з іншими пристроями.

Основні функції:

- Serial.begin(baudrate);
- Serial.write(data);
- Serial.read();

Serial.begin(baudrate) - потрібна для ініціалізації серійного порту з вказаною швидкістю обміну даними. Serial.write(data) - потрібна для відправки даних через серійний порт. Serial.read() - відповідає за зчитування даних із серійного порту.

Ця бібліотека використовується для взаємодії з модулем PZEM004T, який забезпечує вимірювання напруги, струму та потужності. Дані передаються через серійний порт.

```
#include <PZEM004Tv30.h>
```

Бібліотека PZEM004Tv30 використовується для роботи з датчиком PZEM004T, який вимірює напругу, струм, потужність та інші електричні параметри. Ця бібліотека надає простий інтерфейс для зчитування даних з цього датчика.

Основні функції:

- pzem.voltage();
- pzem.current();
- pzem.power();

pzem.voltage() - потрібно для зчитування напруги. pzem.current() - потрібно для зчитування струму. pzem.power() - потрібно для зчитування потужності.

Ця бібліотека дозволяє зчитувати електричні параметри з датчика PZEM004T, що є важливим для моніторингу стану електроживлення в серверній кімнаті.

```
#include <BlynkSimpleEsp32.h>
```

Бібліотека BlynkSimpleEsp32 використовується для інтеграції з платформою Blynk, яка забезпечує віддалений моніторинг та керування пристроями через Інтернет. Ця бібліотека забезпечує всі необхідні функції для підключення до сервера Blynk, відправки та отримання даних.

Основні функції:

- Blynk.begin(auth, ssid, pass);
- Blynk.virtualWrite(pin, value);

- Blynk.run();

Blynk.begin(auth, ssid, pass) - потрібно для підключення до сервера Blynk з вказаними параметрами аутентифікації та мережі. Blynk.virtualWrite(pin, value) - потрібнп нам для відправки даних на віртуальний пін Blynk. Blynk.run() - це основний цикл обробки Blynk.

Ця бібліотека використовується для відправки виміряних параметрів (температура, вологість, напруга, струм тощо) на сервер Blynk, а також для отримання команд для керування пристроєм. Blynk дозволяє легко створювати мобільні додатки для моніторингу та керування IoT-пристроями.

Підключені бібліотеки забезпечують всі необхідні функції для роботи з різними датчиками та модулями, забезпечуючи зручний інтерфейс для взаємодії з ними. Це дозволяє швидко та ефективно розробляти складні проекти на основі платформи ESP32, забезпечуючи широкий спектр функціоналу для моніторингу та керування пристроями через Інтернет. Наступним кроком ми розглянемо кожен з компонентів коду детально, включаючи налаштування пінів, ініціалізацію датчиків, функції для зчитування та відправки даних, а також логіку обробки подій.

2.2 Налаштування апаратних компонентів

Отже наступний важливий крок це налаштування пінів, адже без налаштування, наш прилад, не буде працювати, через не співідношення запрограмовних пінів з реальними:

```
#define DHT11PIN 25
#define MQ2_D 33
#define MQ2_A 34
#define FLAME_D 26
#define FLAME_A 27
#define BUZZER 13
#define TX 16
#define RX 17
```

Ці директиви ‘#define’ визначають відповідність пінів мікроконтролера ESP32 до конкретних функцій або підключених пристроїв. Це дозволяє легко змінювати конфігурацію апаратних з'єднань без необхідності змінювати основний код.

DHT11PIN (пін 25). Пін 25 використовується для зчитування даних з датчика температури та вологості DHT11. DHT11 є цифровим датчиком, який підключається до одного піну для передачі даних про температуру та вологість. Цей датчик має вбудований резистор для вимірювання температури та вологості повітря, що робить його простим у використанні та надійним. Він передає дані у цифровому форматі, що забезпечує високу точність та стабільність вимірювань. Для порівняння, аналогові датчики температури, такі як LM35, вимагають перетворення аналогового сигналу в цифровий, що може вносити похибки і складнощі в обробку даних. DHT11 дозволяє уникнути цих проблем завдяки своєму цифровому виходу.

MQ2_D (пін 33) та MQ2_A (пін 34). Піни 33 і 34 використовуються для зчитування даних з газового датчика MQ2. MQ2_D (пін 33) - це пін для цифрового виходу. Він може бути використаний для простого виявлення наявності газу в повітрі. Якщо концентрація газу перевищує встановлений поріг, вихідний сигнал змінюється з низького на високий або навпаки, в залежності від конфігурації. MQ2_A (пін 34) - це пін для аналогового виходу. Він надає точнішу інформацію про концентрацію газу в повітрі. Аналоговий сигнал змінюється в залежності від концентрації газу, що дозволяє отримати більш детальну і точну інформацію про навколишнє середовище. Для порівняння, використання лише цифрового виходу обмежує можливість отримати детальну інформацію про концентрацію газу, тоді як аналоговий вихід дозволяє вимірювати різні рівні концентрації, що може бути важливим для більш точного моніторингу.

FLAME_D (пін 26) та FLAME_A (пін 27). Піни 26 і 27 використовуються для зчитування даних з датчика полум'я. FLAME_D (пін 26) - це пін для

цифрового виходу. Він використовується для визначення наявності полум'я. Якщо датчик виявляє полум'я, сигнал змінюється з низького на високий або навпаки. Це дозволяє швидко реагувати на пожежі, увімкнувши тривогу або інші захисні механізми. FLAME_A (пін 27) - це пін для аналогового виходу. Він дає змогу отримати інформацію про інтенсивність полум'я. Аналоговий вихід дозволяє виміряти рівень світла, що випромінюється полум'ям, що корисно для оцінки розміру і інтенсивності пожежі. Наприклад, при використанні лише цифрового виходу, ми можемо лише знати про наявність або відсутність полум'я, тоді як аналоговий вихід надає детальну інформацію про його інтенсивність.

BUZZER (пін 13). Пін 13 використовується для керування сигналізатором (баззером). Баззер вмикається або вимикається в залежності від умов, наприклад, при виявленні газу або полум'я. Використання баззера дозволяє швидко сповіщати користувача про небезпечні умови. Він може бути налаштований на видачу різних звукових сигналів в залежності від типу небезпеки, що забезпечує ефективне та оперативне інформування. Для порівняння, світлодіодні індикатори можуть бути менш помітними в шумному середовищі, тоді як звуковий сигнал баззера важче проігнорувати.

TX (пін 16) та RX (пін 17). Піни 16 і 17 використовуються для серійного зв’язку з модулем PZEM004T, який вимірює напругу, струм та потужність. TX (пін 16) - це пін передачі даних (Transmit). Він передає дані від мікроконтролера до модуля PZEM004T. RX (пін 17) - це пін прийому даних (Receive). Він приймає дані від модуля PZEM004T до мікроконтролера. Використання серійного зв'язку дозволяє надійно передавати дані між мікроконтролером та модулем вимірювання параметрів електромережі. PZEM004T надає точні дані про напругу, струм та потужність, що дозволяє здійснювати моніторинг електричної системи в режимі реального часу. Для порівняння, використання аналогових методів вимірювання може бути менш точним і вимагати додаткових схем для обробки сигналів.

2.3 Опис програмних функцій, реалізованих в системі

Наступною частинкою коду є ініціалізацію Blynk та налаштування для з'єднання з WiFi. Це дуже важливо, адже це основний функціонал приладу.

```
#define BLYNK_TEMPLATE_ID "TMPL4DlyJdNk2"
#define BLYNK_TEMPLATE_NAME "TopTool"
#define BLYNK_AUTH_TOKEN "TBizLurLbq8zl2rvKGQ2fMbHTcIyvmEs"
#include <BlynkSimpleEsp32.h>
```

Ці рядки коду необхідні для налаштування Blynk, популярної платформи для IoT (Інтернет речей) проектів, що дозволяє віддалено керувати пристроями через інтернет за допомогою мобільного додатку. Blynk надає зручний інтерфейс для моніторингу та керування різними аспектами вашого проекту, такими як вимірювання температури, вологості, напруги тощо.

```
#define BLYNK_TEMPLATE_ID "TMPL4DlyJdNk2"
```

Цей рядок задає унікальний ідентифікатор шаблону Blynk, який був створений для конкретного проекту. Кожен проект в Blynk має свій власний унікальний ідентифікатор шаблону, який визначає структуру проекту, зокрема, які віджети та параметри будуть використовуватися. Це дозволяє Blynk додатку розпізнати та налаштуватися під конкретний проект, відображаючи необхідну інформацію та надаючи можливість керування пристроями.

```
#define BLYNK_TEMPLATE_NAME "TopTool"
```

Це назва шаблону проекту. Вона використовується для ідентифікації проекту в мобільному додатку та на сервері Blynk. Назва проекту повинна бути унікальною та зрозумілою, щоб користувач міг легко розпізнати свій проект серед інших. Назва також допомагає в організації та керуванні кількома проектами одночасно, забезпечуючи чітку ідентифікацію кожного з них.

```
#define BLYNK_AUTH_TOKEN "TBizLurLbq8zl2rvKGQ2fMbHTcIyvmEs"
```

Токен автентифікації для підключення до серверу Blynk. Цей токен унікальний для кожного користувача та проекту і використовується для забезпечення безпечного з'єднання між пристроєм та сервером Blynk. Токен

необхідний для автентифікації пристрою при підключенні до сервера, що гарантує, що дані передаються між авторизованими пристроями та сервером.

```
#include <BlynkSimpleEsp32.h>
```

Ця бібліотека надає функції для роботи з платформою Blynk на основі мікроконтролерів ESP32. Вона дозволяє легко підключитися до сервера Blynk, відправляти та отримувати дані з мобільного додатку. Бібліотека містить всі необхідні інструменти для інтеграції Blynk з ESP32, забезпечуючи простоту використання та функціональність.

```
BlynkTimer timer;
```

BlynkTimer – це спеціальний таймер, який використовується для регулярного виконання певних функцій у програмі. Він є важливим інструментом для розробників, оскільки дозволяє виконувати функції з певним інтервалом часу, що дуже корисно для регулярного зчитування даних з датчиків або відправки даних на сервер. Наприклад, якщо потрібно щосекунди зчитувати показники температури або вологості та відправляти ці дані на сервер для моніторингу в режимі реального часу, BlynkTimer дозволяє це робити без необхідності в складних циклах або ручному управлінні інтервалами.

```
char auth[] = "TBizLurLbq8zl2rvKGQ2fMbHTcIyvmEs";
char ssid[] = "owerwifi";
char pass[] = "vi75ne74sv03zo11";
```

Ці рядки містять налаштування для з'єднання з WiFi мережею, що є критичним для роботи IoT пристрою. Після початкової ініціалізації програми, ESP32 намагається підключитися до вказаної WiFi мережі, використовуючи SSID та пароль. Після успішного з'єднання з WiFi, ESP32 використовує токен автентифікації для підключення до серверу Blynk.

```
char auth[] = "TBizLurLbq8zl2rvKGQ2fMbHTcIyvmEs";
```

Це масив символів, що містить токен автентифікації Blynk, який був визначений раніше. Він використовується для автентифікації при з'єднанні з сервером Blynk. Токен є ключем, який дозволяє серверу Blynk ідентифікувати пристрій та дозволити йому відправляти і отримувати дані. Це важлива частина

безпеки, що гарантує, що лише авторизовані пристрої можуть підключатися до вашого проекту.

```
char ssid[] = "owerwifi";
```

SSID – це ім'я WiFi мережі, до якої потрібно підключитися. Це ім'я задається вашим WiFi роутером і використовується для ідентифікації мережі. Пристрій ESP32 використовує це ім'я для пошуку та підключення до потрібної мережі.

```
char pass[] = "vi75ne74sv03zo11";
```

Пароль для підключення до WiFi мережі. Цей пароль повинен відповідати налаштуванням WiFi роутера. Введення правильного паролю дозволяє пристрою підключитися до мережі та отримати доступ до інтернету. Без правильного паролю пристрій не зможе підключитися до мережі, що зробить неможливим використання всіх онлайн функцій Blynk.

Як це працює разом:

1. Підключення до WiFi - після початкової ініціалізації програми, мікроконтролер ESP32 використовує надані SSID та пароль для підключення до WiFi мережі. Якщо з'єднання успішне, пристрій отримує доступ до інтернету, що є необхідним для подальшого підключення до Blynk сервера. Наявність стабільного інтернет-з'єднання дозволяє пристрою безперервно відправляти та отримувати дані, забезпечуючи постійний моніторинг та управління.

2. Підключення до Blynk - після успішного з'єднання з WiFi, ESP32 використовує токен автентифікації для підключення до сервера Blynk. Це дозволяє пристрою взаємодіяти з мобільним додатком Blynk, відправляти дані від датчиків та отримувати команди від користувача. Підключення до сервера Blynk відкриває широкий спектр можливостей для віддаленого моніторингу та управління пристроєм через інтуїтивно зрозумілий інтерфейс мобільного додатку.

3. Використання таймерів - BlynkTimer використовується для виконання певних функцій через регулярні інтервали часу, наприклад, зчитування даних з датчиків та відправка їх на сервер Blynk. Це забезпечує

автоматизацію та регулярність збирання даних, що є важливим для точного моніторингу стану різних параметрів, таких як температура, вологість, напруга тощо. Регулярне оновлення даних дозволяє своєчасно реагувати на зміни та забезпечувати безпеку та ефективність роботи системи.

Ця частина коду забезпечує основні налаштування для роботи з Blynk та підключення до WiFi, що є основою для подальшої роботи всієї системи. Після цього налаштування можна перейти до ініціалізації датчиків та іншого обладнання, яке буде використовуватися для моніторингу та керування.

Подальшим кроком в написанні логіки для нашого приладу є ініціалізація об'єктів для роботи з датчиками та дисплеєм. Наступа частина коду ініціалізує об'єкти для роботи з рідкокристалічним дисплеєм (LCD), датчиком температури та вологості (DHT11), а також датчиком вимірювання напруги, струму та потужності (PZEM004T).

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
DHT dht(DHT11PIN, DHT11);
PZEM004Tv30 pzem(Serial2, TX, RX);
```

Ініціалізація LCD дисплея - LiquidCrystal_I2C lcd(0x27, 16, 2). Цей рядок коду створює об'єкт lcd для роботи з рідкокристалічним дисплеєм з інтерфейсом I2C. LiquidCrystal_I2C - це клас, що використовується для роботи з рідкокристалічним дисплеєм через інтерфейс I2C. Цей клас спрощує взаємодію з дисплеєм, дозволяючи легко відображати текст та інші дані на екран. 0x27 - це I2C адреса дисплея. Кожен I2C пристрій має свою унікальну адресу, за допомогою якої мікроконтролер взаємодіє з ним. Адреса 0x27 є типовою для багатьох LCD дисплеїв з I2C інтерфейсом. 16, 2 - ці параметри вказують розміри дисплея – 16 символів у ширину та 2 рядки у висоту. Це означає, що дисплей може відображати до 32 символів одночасно, розподілених на два рядки по 16 символів кожен.

Переваги використання I2C LCD дисплея. Менше проводів - I2C інтерфейс використовує лише два проводи (SDA та SCL) для передачі даних, що значно зменшує кількість необхідних підключень порівняно з паралельним

інтерфейсом. Простота коду - бібліотека LiquidCrystal_I2C надає зручний та простий спосіб роботи з дисплеєм, абстрагуючи низькорівневі деталі взаємодії. Розширюваність - I2C шина дозволяє підключати декілька пристроїв до одних і тих же двох проводів, що дозволяє використовувати більше датчиків та інших периферійних пристроїв на одному мікроконтролері.

Далі потрібно ініціалізувати датчик DHT11.

```
DHT dht(DHT11PIN, DHT11);
```

Цей рядок коду створює об'єкт dht для роботи з датчиком температури та вологості DHT11. DHT - це клас, який надає методи для роботи з датчиками серії DHT (DHT11, DHT22 тощо). Він дозволяє легко зчитувати показники температури та вологості. DHT11PIN - це пін мікроконтролера, до якого підключений датчик DHT11. В даному випадку, це пін 25. DHT11 - це вказівка на тип датчика, що використовується. DHT11 є базовим датчиком температури та вологості, який надає дані з певною точністю та інтервалом.

Переваги використання DHT11. Простота використання - бібліотека DHT забезпечує легкий спосіб зчитування даних з датчика, абстрагуючи всі складні аспекти роботи з протоколом передачі даних. Дешевизна - DHT11 є недорогим датчиком, що робить його доступним для багатьох проектів. Компактність - датчик має невеликі розміри, що дозволяє його використовувати в компактних пристроях та системах.

Ініціалізація датчика PZEM004Tv30.

```
PZEM004Tv30 pzem(Serial2, TX, RX);
```

Цей рядок коду створює об'єкт pzem для роботи з модулем PZEM004Tv30, який вимірює напругу, струм та потужність. PZEM004Tv30 - це клас, що надає методи для роботи з модулем PZEM004Tv30, який вимірює електричні параметри (напруга, струм, потужність). Serial2 - використовується другий апаратний серійний порт мікроконтролера ESP32 для з'єднання з модулем PZEM004Tv30. ESP32 має кілька апаратних серійних портів, що дозволяє підключати кілька серійних пристроїв одночасно. TX та RX - ці піни використовуються для серійної передачі та прийому даних. В даному випадку,

TX - це пін 16, а RX - це пін 17. TX відповідає за передачу даних від мікроконтролера до PZEM004Tv30, а RX - за прийом даних від PZEM004Tv30 до мікроконтролера.

Модуль PZEM004Tv30 забезпечує точні вимірювання напруги, струму та потужності, що важливо для моніторингу енергоспоживання та управління електричними навантаженнями. Простота інтеграції - використання бібліотеки PZEM004Tv30 спрощує процес інтеграції модуля з мікроконтролером, надаючи зручні методи для зчитування даних. Безпека - моніторинг електричних параметрів дозволяє вчасно виявляти проблеми, такі як перевантаження чи нестабільність напруги, що може запобігти пошкодженню обладнання.

Ці три об'єкти (lcd, dht, pzem) забезпечують основну функціональність нашого проекту. Відображення даних на LCD - об'єкт lcd дозволяє виводити на екран поточні значення температури, вологості, напруги, струму та потужності, що зчитуються з відповідних датчиків. Це забезпечує зручний спосіб моніторингу стану системи в реальному часі. Зчитування даних з датчика DHT11 - об'єкт dht використовується для зчитування показників температури та вологості. Ці дані важливі для контролю мікроклімату в приміщенні або в певній зоні, де розташований пристрій. Моніторинг електричних параметрів - об'єкт pzem використовується для вимірювання напруги, струму та потужності. Це необхідно для моніторингу енергоспоживання, що дозволяє виявляти аномалії та оптимізувати використання енергії.

Використання цих об'єктів дозволяє створити інтегровану систему моніторингу, яка забезпечує комплексне збирання даних та їх відображення, а також дозволяє вчасно реагувати на зміни в умовах середовища або електричних параметрів.

Наступна частина коду це - оголошення та ініціалізація змінних.

```
int gas_value = 0;
int gas_status = 0;
float humi = dht.readHumidity();
float temp = dht.readTemperature();
int flame_status = 0;
```

```
int flame_value = 0;
float W = 0;
float voltage = 0;
float current = 0;
```

Цей блок коду оголошує та ініціалізує змінні, які використовуються для зберігання показників датчиків та вимірювань. Ці змінні будуть використовуватись у програмі для зчитування, обробки та відображення даних. Давайте розглянемо кожну змінну окремо.

Змінні для газового датчика MQ2.

```
int gas_value = 0;
```

gas_value - це змінна типу int (ціле число), яка зберігає аналогові значення, зчитані з газового датчика MQ2. Тип даних - тип int використовується для зберігання цілих чисел. Це підходить для значень, зчитаних з аналогового входу, які знаходяться в діапазоні від 0 до 1023 (для 10-бітного АЦП). Аналогове зчитування - газовий датчик MQ2 зчитує концентрацію газу (наприклад, метану, пропану) в повітрі і видає аналоговий сигнал, який пропорційний концентрації газу. Це значення перетворюється на цифровий код АЦП (аналого-цифрового перетворювача), що і зберігається в змінній gas_value. Ініціалізація - початкове значення змінної встановлюється в 0, щоб уникнути некоректних даних до першого зчитування з датчика.

```
int gas_status = 0;
```

gas_status - це змінна типу int, яка зберігає цифрове значення, зчитане з газового датчика MQ2. Тип даних - тип int використовується для зберігання цілих чисел, але в даному випадку значення буде або 0, або 1 (логічне значення). Цифрове зчитування - газовий датчик MQ2 може також видавати цифровий сигнал, який вказує на наявність або відсутність газу вище певного порогу. Це значення зчитується і зберігається в змінній gas_status. Ініціалізація - початкове значення змінної встановлюється в 0, що означає відсутність газу за замовчуванням.

Змінні для датчика температури та вологості DHT11.

```
float humi = dht.readHumidity();
```

humi - це змінна типу float, яка зберігає значення вологості, зчитане з датчика DHT11.Тип даних - тип float використовується для зберігання чисел з плаваючою комою, що дозволяє отримати точніші значення вологості. Зчитування вологості - метод dht.readHumidity() зчитує поточну відносну вологість з датчика DHT11 і повертає значення у відсотках (%). Це значення зберігається в змінній humi. Змінна humi ініціалізується одразу зчитаним значенням вологості. Це дозволяє відразу мати актуальне значення для подальшої роботи програми.

```
float temp = dht.readTemperature();
```

temp - це змінна типу float, яка зберігає значення температури, зчитане з датчика DHT11. Тип даних - тип float дозволяє зберігати точні значення температури з плаваючою комою. Зчитування температури - метод dht.readTemperature() зчитує поточну температуру з датчика DHT11 і повертає значення в градусах Цельсія. Це значення зберігається в змінній temp. Змінна temp ініціалізується одразу зчитаним значенням температури, що забезпечує актуальність даних з моменту запуску програми.

```
int flame_status = 0;
```

flame_status - це змінна типу int, яка зберігає цифрове значення, зчитане з датчика полум'я. Тип даних - тип int використовується для зберігання логічного значення, яке може бути або 0 (немає полум'я), або 1 (є полум'я). Датчик полум'я видає цифровий сигнал, що вказує на наявність або відсутність полум'я. Це значення зчитується і зберігається в змінній flame_status. Початкове значення встановлюється в 0, що означає відсутність полум'я за умовчанням.

```
int flame_value = 0;
```

flame_value - це змінна типу int, яка зберігає аналогове значення, зчитане з датчика полум'я. Тип даних - тип int підходить для зберігання значень, зчитаних з аналогового входу, які знаходяться в діапазоні від 0 до 1023. Датчик полум'я може видавати аналоговий сигнал, який пропорційний інтенсивності полум'я. Це значення зчитується і зберігається в змінній flame_value. Початкове значення

змінної встановлюється в 0, щоб уникнути некоректних даних до першого зчитування з датчика.

```
float W = 0;
```

W - це змінна типу float, яка зберігає значення потужності, виміряне модулем PZEM004T. Тип даних float дозволяє зберігати числові значення з плаваючою комою для більшої точності вимірювань. Модуль PZEM004T вимірює електричну потужність і повертає значення, яке зберігається в змінній W. Початкове значення встановлюється в 0, щоб уникнути некоректних даних до першого вимірювання.

```
float voltage = 0;
```

voltage - це змінна типу float, яка зберігає значення напруги, виміряне модулем PZEM004T. Тип дпних float використовується для зберігання числових значень з плаваючою комою, що дозволяє отримати точніші вимірювання напруги. Модуль PZEM004T вимірює електричну напругу і повертає значення, яке зберігається в змінній voltage. Початкове значення встановлюється в 0, щоб уникнути некоректних даних до першого вимірювання.

```
float current = 0;
```

current - це змінна типу float, яка зберігає значення струму, виміряне модулем PZEM004T. Тип даних float дозволяє зберігати точні значення струму, виміряного модулем PZEM004T. Модуль PZEM004T вимірює електричний струм і повертає значення, яке зберігається в змінній current. Початкове значення встановлюється в 0, щоб уникнути некоректних даних до першого вимірювання.

Цей блок коду готує змінні для зберігання важливих даних, зчитаних з різних датчиків. Кожна змінна ініціалізується початковим значенням, що забезпечує коректність роботи програми з самого початку. Змінні використовуються для подальшої обробки та відображення даних, що робить систему інтегрованою та функціональною.

Наступний крок, це створення функції setup:

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

```
  dht.begin();
  lcd.init();
  lcd.backlight();
  Blynk.begin(auth, ssid, pass);


  pinMode(MQ2_A, INPUT);
  pinMode(MQ2_D, INPUT);
  pinMode(FLAME_A, INPUT);
  pinMode(FLAME_D, INPUT);
  pinMode(BUZZER, OUTPUT);

  timer.setInterval(1000L, sendUptime);
}
```

Функція setup() є однією з ключових функцій в програмуванні на платформі Arduino. Вона виконується один раз при запуску або перезавантаженні мікроконтролера. В цій функції налаштовуються параметри, ініціалізуються компоненти та встановлюються режими роботи пінів. Розглянемо кожен рядок коду всередині setup() детально. Функція setup() оголошується без параметрів та не повертає значень (void). Ця функція автоматично викликається один раз після того, як мікроконтролер перезавантажується або увімкнений.

```
Serial.begin(115200);
```

Цей рядок ініціалізує послідовний зв'язок між мікроконтролером та комп'ютером. Аргумент 115200 встановлює швидкість передачі даних в бодах. Це одна з популярних швидкостей, яка забезпечує швидку та стабільну передачу даних. Серійний зв'язок використовується для налагодження та моніторингу. Ви можете виводити дані на серійний монітор Arduino IDE, щоб бачити, як працює ваш код та які дані зчитуються з датчиків.

```
dht.begin();
```

Цей рядок коду ініціалізує датчик DHT11. Цей рядок викликає метод begin() для об'єкта dht, що ініціалізує датчик DHT11 для зчитування температури та вологості. Метод begin() налаштовує внутрішні параметри датчика та готує

його до роботи. Після виклику цього методу датчик готовий до зчитування даних про температуру та вологість.

```
lcd.init();
```

lcd.init() - це функція ініціалізації LCD-дисплея, який підключений до мікроконтролера через інтерфейс I2C. Ця функція викликається для того, щоб налаштувати дисплей і підготувати його до роботи. I2C (Inter-Integrated Circuit) - це послідовний комунікаційний протокол, який дозволяє підключити до мікроконтролера декілька периферійних пристроїв, використовуючи лише два проводи, SDA (дані) і SCL (годинник). Цей інтерфейс є дуже зручним для підключення дисплеїв, сенсорів та інших модулів, зменшуючи кількість потрібних пінів. У нашому коді об'єкт lcd налаштований для роботи з дисплеєм, що має адресний простір 0x27 (як вказано при створенні об'єкта: LiquidCrystal_I2C lcd(0x27, 16, 2)). Бібліотека LiquidCrystal_I2C абстрагує складність низькорівневих команд, необхідних для керування LCD-дисплеєм через інтерфейс I2C. Вона надає простий у використанні API для ініціалізації та роботи з дисплеєм. Бібліотека підтримує різні розміри дисплеїв (наприклад, 16x2, 20x4) і дозволяє легко відображати текст та символи на дисплеї. Функція init() виконує ряд операцій для налаштування дисплея. Встановлює початкові значення регістрів. Вибирає режим роботи (4-бітний або 8-бітний). Вмикає дисплей та підсвітку. Очищує дисплей і переміщує курсор у верхній лівий кут. Функція init() надсилає послідовність команд до дисплея, щоб налаштувати його. Це включає команди для вибору режиму роботи, налаштування рядків та стовпців, а також для включення або вимкнення певних функцій (наприклад, курсора, підсвітки). Після ініціалізації дисплей очищується, і курсор переміщується у верхній лівий кут. Це забезпечує готовність дисплея до відображення нових даних. Функція встановлює режим роботи дисплея (наприклад, 4-бітний режим передачі даних) для оптимізації роботи з мікроконтролером. Після виклику lcd.init(), дисплей готовий до використання. Ви можете відразу почати відображати текст та іншу інформацію на ньому. Також правильна ініціалізація забезпечує стабільну роботу дисплея, що є

критично важливим для надійності всієї системи. Використання бібліотеки LiquidCrystal_I2C та її функції init() спрощує процес роботи з дисплеєм, звільняючи нас від необхідності писати складний низькорівневий код. Існують інші способи, ініціалізація LCD-дисплея без використання спеціалізованої бібліотеки, проте це може бути дуже складно і вимагати значно більше коду, оскільки нам доведеться вручну надсилати всі необхідні команди та налаштовувати регістри нашого дисплея. Також існують інші бібліотеки для роботи з LCD-дисплеями (наприклад, Adafruit_LiquidCrystal), але LiquidCrystal_I2C є популярною, простою, дуже зручною та має хорошу документацію. Отж lcd.init() є важливою частиною процесу ініціалізації LCD-дисплея, що забезпечує його готовність до роботи. Використання цієї функції разом з бібліотекою LiquidCrystal_I2C значно спрощує нам роботу з дисплеєм, роблячи її більш ефективною і менш схильною до помилок.

```
lcd.backlight();
```

lcd.backlight() - це функція, яка вмикає підсвітку LCD-дисплея. Ця функція є частиною бібліотеки LiquidCrystal_I2C і використовується для керування підсвіткою дисплея, що робить його більш читабельним в умовах низької освітленості. Підсвітка (backlight) є важливою частиною LCD-дисплеїв, оскільки вона дозволяє користувачеві чітко бачити текст та зображення на дисплеї навіть в умовах низької освітленості. Підсвітка зазвичай реалізується за допомогою світлодіодів (LED), розміщених позаду або по боках дисплея. Бібліотека LiquidCrystal_I2C надає функцію backlight(), яка дозволяє легко вмикати або вимикати підсвітку дисплея. Використання цієї бібліотеки робить керування підсвіткою зрозумілим і доступним. Функція backlight() викликається для ввімкнення підсвітки дисплея. Вона взаємодіє з контролером підсвітки через інтерфейс I2C, зазвичай ця функція встановлює відповідний пін контролера в високий логічний рівень, що призводить до ввімкнення світлодіодів підсвітки. Коли викликається lcd.backlight(), бібліотека надсилає відповідну команду на контролер дисплея через шину I2C. Контролер дисплея отримує команду і вмикає світлодіоди підсвітки, роблячи дисплей більш читабельним. Ввімкнення

підсвітки збільшує енергоспоживання дисплея, тому в деяких випадках може бути доцільно вимикати підсвітку, коли вона не потрібна, щоб зберегти енергію. Ввімкнення підсвітки значно покращує видимість інформації на дисплеї, особливо в умовах поганої освітленості або в темряві. Можливість керувати підсвіткою через простий виклик функції (lcd.backlight()) робить користування дисплеєм більш зручним. Ввімкнення підсвітки збільшує споживання енергії, що може бути критичним для пристроїв з живленням від батарей. Також світлодіоди споживають відносно мало енергії, їх постійне використання може призвести до додаткового тепловиділення, що може вплинути на роботу інших компонентів у дуже компактних пристроях.

```
Blynk.begin(auth, ssid, pass);
```

Blynk.begin(auth, ssid, pass) - це функція з бібліотеки Blynk, яка ініціалізує з'єднання між мікроконтролером (в даному випадку ESP32) і платформою Blynk через Wi-Fi. Ця функція забезпечує підключення до Wi-Fi мережі та встановлення зв'язку з сервером Blynk для подальшого обміну даними. Blynk - це популярна платформа для Інтернету речей (IoT), яка дозволяє легко створювати додатки для управління та моніторингу різноманітних пристроїв через Інтернет. Бібліотека Blynk надає простий спосіб інтеграції з цією платформою, забезпечуючи функції для з'єднання, передачі даних і керування пристроями. Функція begin() є ключовою для встановлення з'єднання між мікроконтролером і платформою Blynk. Вона приймає три параметри: auth, ssid і pass. auth - це токен аутентифікації, який видається платформою Blynk для кожного проекту. Він унікальний для кожного проекту і використовується для ідентифікації пристрою на сервері Blynk. Токен аутентифікації забезпечує безпеку з'єднання і доступ до нашого проекту. У нашому випадку, змінна auth містить значення "TBizLurLbq8zl2rvKGQ2fMbHTcIyvmEs", яке ми отримали під час створення проекту в додатку Blynk. ssid - це назва Wi-Fi мережі, до якої має підключитися мікроконтролер. SSID (Service Set Identifier) - це унікальне ім'я, що ідентифікує бездротову мережу. У нашому випадку, змінна ssid містить значення "owerwifi", що є іменем Wi-Fi мережі мого друга з яким ми розробляли

цей проект. pass - це пароль для доступу до Wi-Fi мережі. Пароль забезпечує безпеку уашої мережі, запобігаючи несанкціонованому доступу. У нашому випадку, змінна pass містить значення "vi75ne74sv03zo11", що є паролем нашої Wi-Fi мережі. Функція begin() спочатку ініціалізує Wi-Fi модуль на мікроконтролері ESP32, використовуючи вказані SSID і пароль для підключення до бездротової мережі. Мікроконтролер намагається підключитися до вказаної Wi-Fi мережі. Це може зайняти деякий час, в залежності від сили сигналу та інших факторів. Після успішного підключення до Wi-Fi мережі, функція begin() встановлює з'єднання з сервером Blynk, використовуючи токен аутентифікації для ідентифікації пристрою. Після встановлення з'єднання, мікроконтролер може обмінюватися даними з платформою Blynk. Це включає в себе відправку даних з сенсорів, отримання команд від користувача через мобільний додаток Blynk і багато іншого. У нашому проекті цей рядок забезпечує підключення до платформи Blynk, що дозволяє віддалено контролювати та моніторити дані з різних датчиків, таких як DHT11 (температура і вологість), датчик полум'я, газовий датчик MQ2 і датчик напруги PZEM004T. Через платформу Blynk ми можемо налаштовувати сповіщення, переглядати дані в реальному часі та віддалено керувати пристроями. Рядок Blynk.begin(auth, ssid, pass); є критично важливим для вашого проекту, оскільки він встановлює зв'язок між мікроконтролером ESP32 і платформою Blynk через Wi-Fi мережу. Цей зв'язок дозволяє передавати дані з датчиків на сервер Blynk та отримувати команди від користувача, забезпечуючи повний контроль над вашим IoT-пристроєм. Ініціалізація Wi-Fi і встановлення з'єднання з сервером Blynk за допомогою цієї функції робить можливим використання всіх можливостей платформи Blynk для віддаленого моніторингу та керування.

```
pinMode(MQ2_A, INPUT);
pinMode(MQ2_D, INPUT);
pinMode(FLAME_A, INPUT);
pinMode(FLAME_D, INPUT);
pinMode(BUZZER, OUTPUT);
```

Функція pinMode() використовується для встановлення режиму роботи цифрових пінів мікроконтролера. Вона приймає два параметри: номер піна та режим роботи. Режим може бути INPUT, OUTPUT, або INPUT_PULLUP. pinMode(MQ2_A, INPUT), MQ2_A - це пін мікроконтролера, до якого підключений аналоговий вихід газового датчика MQ2. INPUT - Встановлює режим роботи піна як вхідний (input). Це означає, що цей пін буде використовуватися для зчитування даних з датчика. Аналоговий вихід датчика MQ2 надає значення напруги, пропорційне концентрації газу в повітрі. Пін MQ2_A, налаштований як вхідний, дозволяє мікроконтролеру зчитувати ці значення. pinMode(MQ2_D, INPUT), MQ2_D - це пін мікроконтролера, до якого підключений цифровий вихід газового датчика MQ2. INPUT -встановлює режим роботи піна як вхідний (input). Цифровий вихід датчика MQ2 видає високий або низький сигнал в залежності від концентрації газу, перевищуючи певний поріг. Пін MQ2_D, налаштований як вхідний, дозволяє мікроконтролеру зчитувати цей цифровий сигнал, що інформує про наявність або відсутність газу. pinMode(FLAME_A, INPUT), FLAME_A - це пін мікроконтролера, до якого підключений аналоговий вихід датчика полум'я. INPUT - Встановлює режим роботи піна як вхідний (input). Призначення - аналоговий вихід датчика полум'я надає значення напруги, пропорційне інтенсивності полум'я. Пін FLAME_A, налаштований як вхідний, дозволяє мікроконтролеру зчитувати ці значення. pinMode(FLAME_D, INPUT), FLAME_D - це пін мікроконтролера, до якого підключений цифровий вихід датчика полум'я. Призначення - цифровий вихід датчика полум'я видає високий або низький сигнал в залежності від наявності полум'я. Пін FLAME_D, налаштований як вхідний, дозволяє мікроконтролеру зчитувати цей цифровий сигнал, що інформує про наявність або відсутність полум'я. pinMode(BUZZER, OUTPUT), BUZZER - це пін мікроконтролера, до якого підключений баззер (звуковий сигналізатор). OUTPUT - встановлює режим роботи піна як вихідний (output). Це означає, що мікроконтролер буде використовувати цей пін для керування підключеним до нього пристроєм.

Призначення - вихідний режим для піна BUZZER дозволяє мікроконтролеру включати або вимикати баззер, подаючи високий або низький сигнал на цей пін. Це використовується для звукового сповіщення у разі виявлення полум'я або підвищеної концентрації газу. Кожен з цих рядків коду виконує критичну функцію в налаштуванні мікроконтролера для взаємодії з підключеними датчиками і виконавчими пристроями. Вхідні піни (INPUT) налаштовуються для зчитування даних з датчиків (аналогових і цифрових виходів), тоді як вихідний пін (OUTPUT) налаштовується для керування баззером. Це налаштування забезпечує коректну роботу мікроконтролера в вашій системі, дозволяючи зчитувати необхідні дані та відповідно реагувати на виявлені події.

```
timer.setInterval(1000L, sendUptime);
```

Функція setInterval є методом об'єкта BlynkTimer (який схожий на SimpleTimer) і використовується для встановлення інтервалу таймера. Ця функція виконує певну дію через заданий інтервал часу, що дозволяє повторювати виконання певної функції без використання затримок (delay). Перший параметр функції - 1000L. Це перший параметр функції, який визначає інтервал часу в мілісекундах. Значення 1000L означає, що функція буде виконуватися кожну секунду (1000 мілісекунд). Суфікс L вказує на те, що це довге ціле число (long). У цьому контексті це не обов'язково, але іноді використовується для ясності та уникнення можливих помилок. Наступний – sendUptime. Це другий параметр, який визначає функцію, що буде викликатися через заданий інтервал часу. У цьому випадку, функція sendUptime буде викликатися кожну секунду. Функція sendUptime повинна бути визначена в коді раніше або пізніше, але до її використання, і містити логіку, яку ви хочете виконувати періодично. Як це працює ? Коли ми викликаємо timer.setInterval(1000L, sendUptime), ми створюємо таймер, який викликає функцію sendUptime кожну секунду. Це дозволяє виконувати регулярні дії, наприклад, зчитування даних з датчиків, оновлення дисплея, або відправку даних на сервер без використання затримок. Використання таймера замість затримок (delay) дозволяє мікроконтролеру виконувати інші завдання в

основному циклі (loop()) без блокування. Перевагами використання таймера є плавна робота мікроконтролера. Використання таймерів забезпечує більш плавну роботу програми, оскільки інші функції можуть виконуватися паралельно, поки таймер чекає завершення інтервалу. Наступною перевагою є ефективне управління часом. Таймери дозволяють ефективно управляти часом виконання функцій, забезпечуючи регулярне виконання завдань у визначені моменти часу. Також потрібно зазначити уникнення блокувань. Використання setInterval допомагає уникнути блокування основного циклу програми, яке може виникнути при використанні функції delay. У нашому проекті функція sendUptime може містити логіку зчитування даних з датчиків, оновлення дисплея або відправку даних на сервер Blynk. Виклик sendUptime кожну секунду забезпечує регулярне оновлення даних і своєчасне реагування на зміни. Рядок timer.setInterval(1000L, sendUptime); є важливою частиною нашого коду, що дозволяє виконувати певні дії через регулярні інтервали часу. Використання таймерів замість затримок забезпечує ефективніше і плавніше виконання програми, дозволяючи мікроконтролеру виконувати інші завдання паралельно з основними функціями.

Наступною частиною коду є сама функція sendUptime(), яка описувалася раніше:

```
void sendUptime() {

  voltage = pzem.voltage();
  current = pzem.current();
  W = pzem.power();
  gas_value = analogRead(MQ2_A);
  gas_status = digitalRead(MQ2_D);
  humi = dht.readHumidity();
  temp = dht.readTemperature();
  flame_status = digitalRead(FLAME_D);
  flame_value = analogRead(FLAME_A);

  Blynk.virtualWrite(V5, gas_value);
  Blynk.virtualWrite(V1, temp);
  Blynk.virtualWrite(V2, humi);
```

```
  Blynk.virtualWrite(V3, voltage);
  Blynk.virtualWrite(V4, current);


  if (gas_value > 160) {
    Blynk.logEvent("gas_leaked");
  }
  if (temp > 45.00) {
    Blynk.logEvent("temperature_is_high");
  }
  if (temp < -5.00) {
    Blynk.logEvent("temperature_is_low");
  }
  if (flame_status == 1) {
    Blynk.logEvent("flame");
    digitalWrite(BUZZER, HIGH);
  }
  if (flame_status == 0) digitalWrite(BUZZER, LOW);

  lcd.setCursor(0, 0);
  lcd.print("T:");
  lcd.print(temp);
  lcd.print("C ");
  lcd.print("V:");
  lcd.print(voltage);
  lcd.setCursor(0, 1);
  lcd.print("H:");
  lcd.print(humi);
  lcd.print("% ");
  lcd.print("A:");
  lcd.print(current);

}
```

Розглянемо детально функцію sendUptime() і кожен її рядок коду. Функція sendUptime() виконує зчитування даних з датчиків, відправку цих даних на платформу Blynk і виведення їх на дисплей.

```
voltage = pzem.voltage();
current = pzem.current();
W = pzem.power();
```

voltage = pzem.voltage(). Цей рядок коду зчитує поточне значення напруги (Voltage) з датчика PZEM004T. Викликається метод voltage() об'єкта pzem, який є екземпляром класу PZEM004Tv30. Метод voltage() повертає значення напруги в вольтах. Клас PZEM004Tv30 реалізує комунікацію з датчиком через інтерфейс UART. Метод voltage() відправляє відповідний запит на датчик, який повертає поточне значення напруги. Зчитане значення присвоюється змінній voltage, яку можна використовувати для подальшої обробки або відображення. current = pzem.current(), цей рядок коду зчитує поточне значення струму (Current) з датчика PZEM004T. Викликається метод current() об'єкта pzem, який повертає значення струму в амперах. Метод current() відправляє запит на датчик PZEM004T, який повертає поточне значення струму. Зчитане значення присвоюється змінній current. w = pzem.power(), цей рядок коду зчитує поточне значення потужності (Power) з датчика PZEM004T. Викликається метод power() об'єкта pzem, який повертає значення потужності в ватах. Метод power() відправляє запит на датчик PZEM004T, який повертає поточне значення потужності. Зчитане значення присвоюється змінній w. Методи voltage(), current(), і power() в класі PZEM004Tv30 реалізовані таким чином, що вони відправляють відповідні команди на датчик через послідовний інтерфейс (UART). Датчик обробляє ці команди і відповідає зчитаними значеннями. Ці три рядки коду відіграють важливу роль в зчитуванні даних з датчика PZEM004T. Вони забезпечують отримання значень напруги, струму та потужності, які потім будуть використані для моніторингу, аналізу та відображення стану електричної системи.

```
gas_value = analogRead(MQ2_A);
gas_status = digitalRead(MQ2_D);
```

gas_value = analogRead(MQ2_A). Цей рядок коду зчитує аналогове значення з газового датчика MQ2 через пін MQ2_A. Викликається функція analogRead(), яка зчитує аналоговий сигнал з визначеного нами піну (MQ2_A). Це значення є пропорційним до концентрації газу в повітрі, що вимірюється датчиком MQ2. Працює це наступним чином, газовий датчик MQ2 генерує

аналоговий сигнал, який змінюється в залежності від концентрації газу в повітрі. Функція analogRead() зчитує цей сигнал і повертає значення від 0 до 1023, яке відповідає рівню концентрації газу. Далі зчитане значення присвоюється змінній gas_value. Функція analogRead() працює з 10-бітним АЦП (аналогово-цифровим перетворювачем) мікроконтролера, який перетворює аналоговий сигнал (0-5В для стандартного Arduino) у цифрове значення (0-1023), яке нам потрібно. Значення 0 відповідає 0В на аналоговому піні, а 1023 відповідає максимально можливій напрузі (зазвичай 5В або 3.3В, залежно від плати). gas_status = digitalRead(MQ2_D). Цей рядок коду зчитує цифровий сигнал з газового датчика MQ2 через пін MQ2_D. Викликається функція digitalRead(), яка зчитує цифровий сигнал з визначеного нами піну (MQ2_D). Це значення може бути або HIGH, або LOW, в залежності від порогового значення концентрації газу. Газовий датчик MQ2 може бути налаштований на вивід цифрового сигналу, який змінюється між HIGH і LOW в залежності від того, чи перевищує концентрація газу заданий нами поріг. Функція digitalRead() зчитує цей сигнал і повертає відповідне значення (HIGH або LOW). Ну і далі зчитане значення присвоюється змінній gas_status. Функція digitalRead() зчитує логічний рівень на вказаному піні. Значення HIGH відповідає напрузі, близькій до живлення (наприклад, 3.3В або 5В), а LOW відповідає напрузі, близькій до 0В. Цифровий вихід датчика може бути налаштований за допомогою потенціометра на самому датчику, який встановлює поріг спрацьовування. Ці два рядки коду забезпечують отримання даних з газового датчика MQ2. Перший рядок зчитує аналогове значення, яке дає більш точну інформацію про концентрацію газу, тоді як другий рядок зчитує цифровий сигнал, який використовується для простого виявлення наявності або відсутності газу на основі заданого порогу. Ці дані можуть бути використані для моніторингу якості повітря і виявлення потенційно небезпечних концентрацій газу.

Далі розгляному рядки коду, які зчитують дані з датчика температури та вологості.

```
humi = dht.readHumidity();
```

```
temp = dht.readTemperature();
```

humi = dht.readHumidity(). Цей рядок коду зчитує значення вологості з датчика DHT11. Викликається метод readHumidity() об'єкта dht, який ініціалізований для роботи з датчиком DHT11. Цей метод повертає значення відносної вологості в процентах (%). Датчик DHT11 використовує вбудований сенсор для вимірювання вологості повітря. Метод readHumidity() ініціює процес вимірювання і отримує значення вологості, яке потім повертає, потім зчитане значення присвоюється змінній humi. Метод readHumidity() здійснює запит до датчика і зчитує цифрове значення вологості, яке обробляється і перетворюється у відсотки. Якщо вимірювання невдале, метод може повернути спеціальне значення, яке вказує на помилку. Далі, temp = dht.readTemperature(). Цей рядок коду зчитує значення температури з датчика DHT11. Викликається метод readTemperature() об'єкта dht, який ініціалізований для роботи з датчиком DHT11. Цей метод повертає значення температури в градусах Цельсія (°C). Працює це наступним чином, датчик DHT11 використовує вбудований сенсор для вимірювання температури. Метод readTemperature() ініціює процес вимірювання і отримує значення температури, яке потім повертає і після чого, зчитане значення присвоюється змінній temp. Метод readTemperature() здійснює запит до датчика і зчитує цифрове значення температури, яке обробляється і перетворюється у градуси Цельсія. Як і у випадку з вимірюванням вологості, метод може повернути спеціальне значення у разі невдалого вимірювання або помилки. Ці два рядки коду забезпечують отримання даних про вологість та температуру з датчика DHT11. Перший рядок зчитує відносну вологість у відсотках, тоді як другий рядок зчитує температуру в градусах Цельсія. Ці дані можуть бути використані для моніторингу умов навколишнього середовища, автоматизації систем вентиляції та кондиціонування повітря, а також для інших додатків, що потребують контролю температури і вологості.

Далі розглянемо строки коду, які зчитуютьдані з датчика полум’я.

```
flame_status = digitalRead(FLAME_D);
flame_value = analogRead(FLAME_A);
```

flame_status = digitalRead(FLAME_D). Цей рядок коду зчитує цифрове значення з датчика полум'я, підключеного до піну FLAME_D. Викликається функція digitalRead(), яка зчитує стан піну FLAME_D. Ця функція повертає HIGH або LOW, залежно від наявності полум'я. Датчик полум'я містить інфрачервоний приймач, який виявляє наявність полум'я. Коли полум'я присутнє, на виході датчика формується високий рівень сигналу (HIGH), коли полум'я відсутнє - низький (LOW). Метод digitalRead() зчитує цей сигнал і повертає відповідне значення. Наступним кроком, зчитане значення присвоюється змінній flame_status. Функція digitalRead() зчитує рівень напруги на піну FLAME_D, де HIGH вказує на наявність полум'я, а LOW - на його відсутність. Це дозволяє мікроконтролеру швидко реагувати на виявлення полум'я і виконувати відповідні дії. Наступна зміна flame_value = analogRead(FLAME_A). Даний рядок коду зчитує аналогове значення з датчика полум'я, підключеного до піну FLAME_A. Його призначення, викликати функцію analogRead(), яка зчитує значення напруги на піну FLAME_A. Ця функція повертає значення від 0 до 4095 (для ESP32), що відповідає рівню інтенсивності полум'я. Далі датчик полум'я генерує аналоговий сигнал, який змінюється залежно від інтенсивності полум'я. Після чого метод analogRead() зчитує цей сигнал і повертає значення, яке пропорційне напрузі на піну. І зчитане значення присвоюється змінній flame_value. Функція analogRead() здійснює перетворення аналогового сигналу на цифровий і повертає значення від 0 до 4095, де 0 відповідає 0 В, а 4095 - максимально можливій напрузі (зазвичай 3.3 В). Це значення дозволяє оцінити інтенсивність полум'я, що може бути використано для більш точного моніторингу або регулювання систем безпеки. Ці два рядки коду забезпечують отримання даних з датчика полум'я: flame_status вказує на наявність або відсутність полум'я, тоді як flame_value дозволяє оцінити інтенсивність полум'я. Таке зчитування даних дає можливість швидко реагувати на небезпеку пожежі та вживати відповідних заходів для забезпечення безпеки.

Далі розберемо частину коду, яка відповідає за взаємодію з Blynk.

```
Blynk.virtualWrite(V5, gas_value);
Blynk.virtualWrite(V1, temp);
Blynk.virtualWrite(V2, humi);
Blynk.virtualWrite(V3, voltage);
Blynk.virtualWrite(V4, current);
```

Давайте розберемо кожен рядочок окремо. Blynk.virtualWrite(V5, gas_value). Цей рядок відповідає за відправку даних про концентрацію газу, зчитаних з датчика MQ2, на сервер Blynk. Віртуальний пін V5 використовується для передачі цього значення. Функція Blynk.virtualWrite() відправляє значення змінної gas_value до віртуального піну V5. Це дозволяє відобразити дані в мобільному додатку Blynk або використовувати їх для подальшої обробки. Віртуальні піни в Blynk використовуються для передачі даних між мікроконтролером і сервером Blynk. Вони дозволяють організувати обмін даними без прив'язки до конкретних фізичних пінів. Blynk.virtualWrite(V1, temp). Цей рядок коду призначений для відправки даних про температуру, зчитаних з датчика DHT11, на сервер Blynk. Віртуальний пін V1 використовується для передачі цього значення. Функція Blynk.virtualWrite() відправляє значення змінної temp до віртуального піну V1. Це значення може бути використане для моніторингу температури через мобільний додаток Blynk. Відправка даних через віртуальні піни дозволяє легко інтегрувати різні датчики і пристрої в систему моніторингу та керування на базі Blynk. Blynk.virtualWrite(V2, humi) - цей рядок коду відповідає за відправку даних про вологість, зчитаних з датчика DHT11, на сервер Blynk. Віртуальний пін V2 використовується для передачі цього значення. Його функціонування -функція Blynk.virtualWrite() відправляє значення змінної humi до віртуального піну V2. Це значення може бути використане для моніторингу вологості через мобільний додаток Blynk. Вологість є важливим параметром для моніторингу стану середовища, і передача цих даних на сервер Blynk дозволяє зручно відстежувати зміни в реальному часі. Blynk.virtualWrite(V3, voltage) - цей рядок коду призначений для відправки даних про напругу, зчитаних з вимірювального

модуля PZEM004T, на сервер Blynk. Віртуальний пін V3 використовується для передачі цього значення. Функція Blynk.virtualWrite() відправляє значення змінної voltage до віртуального піну V3. Це значення може бути використане для моніторингу напруги в електричній мережі через мобільний додаток Blynk. Моніторинг напруги є критичним для безпеки та ефективності електричних систем. Передача цих даних на сервер Blynk забезпечує можливість віддаленого контролю та аналізу. Blynk.virtualWrite(V4, current) - цей рядок коду відповідає за відправку даних про струм, зчитаних з вимірювального модуля PZEM004T, на сервер Blynk. Віртуальний пін V4 використовується для передачі цього значення. Функція Blynk.virtualWrite() відправляє значення змінної current до віртуального піну V4. Це значення може бути використане для моніторингу струму в електричній мережі через мобільний додаток Blynk. Моніторинг струму допомагає запобігти перевантаженням та виявити аномальні стани в електричній системі. Передача цих даних на сервер Blynk дозволяє оперативно реагувати на зміни та підтримувати безпеку. Ці рядки коду забезпечують інтеграцію системи моніторингу з платформою Blynk, що дозволяє передавати важливі параметри (концентрація газу, температура, вологість, напруга та струм) на сервер для подальшого аналізу та відображення в мобільному додатку або через web додаток. Це забезпечує користувачеві можливість віддаленого контролю та моніторингу стану системи в реальному часі.

Далі розберемо кілька умов які нам потрібні для коректної і логічної взаємодії з Blynk:

```
if (gas_value > 160) {
   Blynk.logEvent("gas_leaked");
}
```

Цей код реалізує умовну перевірку, щоб визначити, чи перевищує значення газу певний поріг. Поріг у цьому випадку встановлений на рівні 160. Якщо значення газу перевищує цей поріг, виконується код всередині блоку if. Функція analogRead(MQ2_A) раніше в коді зчитала значення з аналогового виводу датчика газу MQ2 і присвоїла його змінній gas_value. Умовний оператор

if перевіряє, чи є значення gas_value більшим за 160. Якщо умова істинна, виконується блок коду всередині фігурних дужок {}. Цей поріг можна налаштувати залежно від вимог системи і чутливості датчика. Встановлення правильного порогу є критичним для забезпечення точного виявлення витоків газу та мінімізації хибних спрацьовувань. Blynk.logEvent("gas_leaked") - якщо умова if виконується, цей рядок коду викликає функцію Blynk.logEvent() для реєстрації події витоку газу. Рядок "gas_leaked" є ідентифікатором події, який може бути налаштований у проекті Blynk для відстеження та реагування на подію витоку газу. Функція Blynk.logEvent() відправляє повідомлення на сервер Blynk про те, що сталася подія, яка відповідає ідентифікатору "gas_leaked". Це повідомлення буде використане для відправки сповіщень користувачу мобільного додатку Blynk або для запуску інших дій, таких як активація сирени чи відправка повідомлення електронною поштою. Використання Blynk.logEvent() дозволяє легко інтегрувати систему сповіщень у проект. Це забезпечує зручний спосіб інформування користувача про критичні події в режимі реального часу. Ідентифікатор події "gas_leaked" повинен бути заздалегідь налаштований у панелі керування Blynk, де можна визначити, які дії мають виконуватись при надходженні цього повідомлення. Цей фрагмент коду відповідає за моніторинг рівня газу та відправку сповіщення про витік газу на сервер Blynk у випадку, якщо концентрація газу перевищує встановлений поріг. Це важлива функція для забезпечення безпеки системи, оскільки своєчасне виявлення витоку газу може запобігти небезпечним ситуаціям і дозволяє оперативно реагувати на інциденти.

Наступна умова:

```
if (temp > 45.00) {
   Blynk.logEvent("temperature_is_high");
}
```

if (temp > 45.00) {}- цей рядок коду виконує перевірку, чи перевищує значення температури певний критичний рівень. У даному випадку, критичним рівнем встановлено значення 45.00 градусів Цельсія. Якщо температура

перевищує цей поріг, виконується блок коду всередині умови if. Значення змінної temp було отримане раніше за допомогою датчика температури і вологості DHT11. Умова if перевіряє, чи є значення temp більшим за 45.00. Якщо ця умова є правдивою, то виконується код всередині фігурних дужок {}. Критичний поріг у 45.00 градусів Цельсія може бути налаштований залежно від конкретних вимог до безпеки або функціональності нашої системи. Важливо встановити правильний поріг для своєчасного виявлення небезпечних ситуацій та запобігання перегріву комплектуючих. Blynk.logEvent("temperature_is_high") викликається, якщо умова if є правдивою, для реєстрації події підвищеної температури. Рядок "temperature_is_high" слугує ідентифікатором події, який використовується для відстеження та реагування на перевищення температури у проекті Blynk. Функція Blynk.logEvent() надсилає повідомлення на сервер Blynk про те, що сталася подія підвищення температури. Це повідомлення буде використане для відправки сповіщень користувачам мобільного додатку Blynk або для виконання інших дій, наприклад, активації охолоджувальних систем чи відправки повідомлення електронною поштою. Використання Blynk.logEvent() забезпечує доступний та ефективний спосіб інтеграції системи сповіщень у проект. Це дозволяє швидко інформувати користувача про критичні події в режимі реального часу. Ідентифікатор події "temperature_is_high" має бути заздалегідь налаштований у панелі керування Blynk, де визначаються відповідні дії на випадок отримання цього повідомлення. Цей фрагмент коду відповідає за моніторинг рівня температури та відправку сповіщення про підвищення температури на сервер Blynk у випадку, якщо температура перевищує встановлений критичний рівень. Це важлива функція для забезпечення безпеки системи, оскільки своєчасне виявлення перегріву може допомогти уникнути небезпечних ситуацій і дозволяє оперативно реагувати на підвищення температури.

Наступною важливою умовою умовою, для контролю температури є:

```
if (temp < -5.00) {
  Blynk.logEvent("temperature_is_low");
```

```
}
```

Ця умова перевіряє, чи значення температури (temp) менше за -5.00 градусів Цельсія. Умовний оператор if дозволяє виконувати певний блок коду тільки тоді, коли умова є істинною. temp - це змінна, яка містить значення температури, зчитане з датчика DHT11. Ця змінна була ініціалізована раніше у функції sendUptime(). `< -5.00` - це порогове значення. Якщо температура менша за це значення, умова стає істинною, і код всередині блоку if виконується. Якщо умова if є істинною, то код буде виконуватися далі і спрацює функція - Blynk.logEvent("temperature_is_low"). Ця функція надсилає подію на сервер Blynk, в якій йдетьсся про те, що температура занизька. logEvent - це метод об'єкта Blynk, який використовується для реєстрації подій. "temperature_is_low" - це назва події. Воно передається як рядковий параметр і використовується для ідентифікації події на сервері Blynk. Тобто якщо температура падає нижче -5.00 градусів Цельсія, виконується код всередині блоку if. І тоді за допомогою методу Blynk.logEvent(), подія з ім'ям "temperature_is_low" надсилається на сервер Blynk. Це дозволяє серверу Blynk зберігати запис про цю подію та, за необхідності, відправляти сповіщення власнику на мобільний додаток чи на веб додаток. Цей блок коду виконує критично важливу функцію моніторингу температури і забезпечує своєчасне сповіщення у тому разі, якщо температура падає до небезпечно низького рівня. Завдяки інтеграції з Blynk, власник отримуватиме сповіщення в реальному часі, що дозволяє оперативно реагувати на зміну умов.

Наступним блоком умов є :

```
if (flame_status == 1) {
   Blynk.logEvent("flame");
   digitalWrite(BUZZER, HIGH);
}
if (flame_status == 0) digitalWrite(BUZZER, LOW);
```

Перший блок if перевіряє, чи значення змінної flame_status дорівнює 1. Якщо ця рівність вірна, тобто повертається true, то це свідчить про виявлення

полум'я датчиком. flame_status - це змінна, яка зберігає стан полум'я, зчитаний з цифрового виходу датчика полум'я. Значення 1 вказує на виявлення полум'я.

Якщо умова if є істинною, виконується цей рядок коду - Blynk.logEvent("flame"). Ця функція надсилає подію на сервер Blynk. logEvent - метод об'єкта Blynk, який використовується для реєстрації подій. "flame" - ім'я події, що передається як рядковий параметр і використовується для ідентифікації події на сервері Blynk. Виклик функції digitalWrite(BUZZER, HIGH), цей рядок коду активує бузер. digitalWrite - функція Arduino, яка встановлює значення цифрового виходу на вказаному піні. BUZZER - пін, до якого підключений бузер. HIGH - високий рівень сигналу (напруга), що активує бузер. Далі йде блок в якому перевряжться умова if (flame_status == 0). Цей рядок перевіряє, чи значення змінної flame_status дорівнює 0. Це свідчить про відсутність полум'я. '== 0' - Оператор порівняння, який перевіряє, чи дорівнює flame_status нулю. Виклик функції digitalWrite(BUZZER, LOW), цей рядок деактивує бузер. LOW - низький рівень сигналу (відсутність напруги), що деактивує бузер. Тобто в цих блоках коду ми перевіряємо, чи не зафіксував наш датчик полум'я. Якщо завіксував то у flame_status записується 1 – це означає, що є вогонь, якщо 0 то все добре. Так як ми з моїм колегою розробляжмо прилад для серверної, то цей блок коду забезпечує архіважливу функцію виявлення полум'я і негайного сповіщення про небезпеку. Якщо ми не будемо слідкувати за можливістю винекнення пожежі, то ризикуємо втратити дороговартісне обладнення а можливо навіть людське життя.

Наступний блок коду:

```
lcd.setCursor(0, 0);
lcd.print("T:");
lcd.print(temp);
lcd.print("C ");
lcd.print("V:");
lcd.print(voltage);
lcd.setCursor(0, 1);
lcd.print("H:");
lcd.print(humi);
```

```
lcd.print("% ");
lcd.print("A:");
lcd.print(current);
```

lcd.setCursor(0, 0) - цей рядок встановлює позицію курсора в нульову колонку (перший символ) та нульовий рядок (перший рядок) дисплея. lcd.print("T:"); - виводить на дисплей текст "T:" для позначення температури. Наступний рядок lcd.print(temp); - виводить на дисплей значення змінної temp, яка містить температуру. lcd.print("C "); - виводить на дисплей символ "C" для позначення одиниць вимірювання температури (Цельсій). lcd.print("V:"); - виводить на дисплей текст "V:" для позначення напруги. lcd.print(voltage); - виводить нам значення змінної voltage, яка містить виміряне значення напруги. lcd.setCursor(0, 1); - встановлює позицію курсора в нульову колонку (перший символ) та перший рядок (другий рядок) дисплея. lcd.print("H:"); - виводить на екран текст "H:" для позначення вологості. lcd.print(humi); - друкує значення змінної humi, яка містить виміряне значення вологості. lcd.print("% "); -виводить на дисплей символ "%" для позначення одиниць вимірювання вологості (відсотки). lcd.print("A:"); - виводить на екран текст "A:" для позначення струму. lcd.print(current); - друкує значення змінної current, яка містить виміряне значення струму. Підсумовуючи можна зрозуміти, що цей блок коду відповідає за відображення основних параметрів (температури, напруги, вологості та струму) на LCD дисплеї. Він спочатку налаштовує курсор на потрібні позиції, а потім виводить відповідні значення змінних разом з текстовими позначеннями та одиницями вимірювання. Це дозволяє користувачеві бачити поточні виміряні параметри системи на дисплеї у зрозумілому форматі.

Далі у розглянемо функцію loop():

```
void loop() {
 Blynk.run();
 timer.run();
}
```

Функція loop() є основною частиною програми на платформі Arduino. Ця функція виконується безперервно в циклі після завершення функції setup(). Вона

забезпечує безперервне виконання коду, що дозволяє мікроконтролеру постійно реагувати на події, обробляти дані та взаємодіяти з користувачем. Blynk.run() - цей рядок забезпечує обробку всіх завдань, пов'язаних з Blynk. Функція Blynk.run() підтримує зв'язок з сервером Blynk. Вона обробляє всі вхідні та вихідні дані, забезпечує актуалізацію віджетів на мобільному додатку Blynk та реагує на події. Завдяки цій функції наш пристрій постійно залишається підключеним до Blynk та може відправляти й отримувати дані в реальному часі. timer.run() - цей рядок забезпечує виконання всіх таймерів, налаштованих в програмі. Функція timer.run() обробляє всі події, пов'язані з таймерами, створеними за допомогою об'єкта BlynkTimer. В даному проекті, таймер використовується для періодичного виклику функції sendUptime() кожну секунду (1000 мілісекунд). Функція timer.run(); перевіряє, чи настав час виконання будь-якого з зареєстрованих таймерів, і виконує відповідні функції, якщо це необхідно. Функція loop() забезпечує безперервну роботу програми, підтримуючи зв'язок з сервером Blynk та керуючи таймерами. Вона гарантує, що ваш пристрій постійно оновлює інформацію на сервері та реагує на зміни у стані датчиків і акторів, забезпечуючи надійну і стабільну роботу всього проекту.

Також важливо сказати і про datastreams, це те, без чого відправка наших даних на хмару Blynk не працюватиме і відстежувати ситуацію в серверній з будь якої іншої точки світу буде не можливо. Датастрім - це канал для передачі даних з пристрою до сервера Blynk. Вони можуть бути використані для відображення даних на графіку, лінійки або іншому візуальному елементі у мобільному чи веб додатку Blynk. Для налаштування датастрімів у Blynk використовується функція Blynk.virtualWrite(). Параметри цієї функції включають номер віртуального піна та значення, яке потрібно передати на сервер Blynk. Наприклад - Blynk.virtualWrite(V1, temp);. У цьому прикладі V1 - це номер віртуального піна у проекті Blynk, а temp - значення температури, яке потрібно відправити на сервер. Тобто дані не просто відсилаються на сервер, а на конкретний, присвоєний певному значенню датастрім. Також

налаштовуються і сповіщення. Blynk надає можливість відправляти сповіщення (notifications) на мобільні пристрої користувачів при виникненні певних подій або станів системи. Для цього використовується функція Blynk.notify(). Приклад використання - Blynk.notify("Gas leaked!");. У цьому прикладі відправляється сповіщення з текстом "Gas leaked!" на всі зареєстровані пристрої користувачів, які підключені до проекту Blynk. У функції sendUptime(), яка викликається з інтервалом часу, ви використовуєте Blynk.virtualWrite() для відправки даних на сервер Blynk для відображення на віджетах у мобільному додатку. Також у випадку певних подій, наприклад, виявлення витоку газу або підвищення температури, ви викликаєте Blynk.logEvent() для реєстрації цих подій на сервері Blynk. Ці події можуть бути налаштовані для відправлення сповіщень користувачам через мобільний додаток Blynk. Налаштовуючи датастріми та сповіщення в Blynk, ми можемо використовувати мобільні додатки, які ефективно візуалізують дані нашого пристрою та повідомляють власника про важливі події в реальному часі. Це дозволяє забезпечити ефективне моніторинг та керування вашим пристроєм з будь-якого місця у світі, де є доступ до Інтернету.

2.4 Висновки по розробці

Цей проект представляє собою комплексну систему моніторингу навколишнього середовища та параметрів електроживлення з використанням мікроконтролера ESP32. Система включає різноманітні датчики для вимірювання температури, вологості, концентрації газу, наявності полум'я та параметрів електроживлення. Отримані дані відображаються на LCD-дисплеї та передаються на сервер Blynk для віддаленого моніторингу та оповіщення користувача про критичні ситуації.

Бібліотеки:

1. LiquidCrystal_I2C.h - бібліотека для роботи з LCD-дисплеєм через інтерфейс I2C.

2. DHT.h - бібліотека для роботи з датчиком температури та вологості DHT11.

3. WiFi.h та WiFiClient.h - бібліотеки для підключення до WiFi мережі.

4. HardwareSerial.h - бібліотека для роботи з апаратним серійним інтерфейсом.

5. PZEM004Tv30.h - бібліотека для роботи з модулем PZEM004T, що вимірює параметри електроживлення.

6. BlynkSimpleEsp32.h - бібліотека для інтеграції з платформою Blynk.

Ініціалізація. setup() - функція виконується один раз при старті програми. Вона налаштовує серійний зв'язок, ініціалізує датчики, дисплей, встановлює режими роботи пінів та підключається до сервера Blynk. Також налаштовується таймер для періодичного виклику функції sendUptime. Основна функціональність. sendUptime() - функція, що викликається кожну секунду. Вона зчитує дані з датчиків, передає їх на сервер Blynk та відображає на LCD-дисплеї. У разі виявлення критичних значень параметрів (висока температура, витік газу, наявність полум'я), функція надсилає відповідні сповіщення через Blynk та активує звуковий сигнал (буззер). Циклічна обробка. loop() - основний цикл програми, який безперервно виконується після ініціалізації. В цьому циклі відбувається обробка подій Blynk та таймерів.

Переваги системи цієї системи:

1. Різноманітність датчиків - прилад використовує кілька датчиків для моніторингу різних параметрів навколишнього середовища та електроживлення.

2. Віддалений моніторинг - завдяки інтеграції з платформою Blynk, користувач може отримувати дані та сповіщення в режимі реального часу на мобільний пристрій.

3. Локальне відображення - дані відображаються на LCD-дисплеї, що дозволяє бачити актуальні значення параметрів безпосередньо на місці встановлення системи.

4. Гнучкість налаштувань - завдяки використанню таймерів та можливості налаштування Blynk, система може бути адаптована під різні вимоги та сценарії використання.

Цей проект демонструє ефективне використання мікроконтролера ESP32 для створення системи моніторингу навколишнього середовища з інтеграцією до інтернету речей (IoT). Він підкреслює важливість використання відповідних бібліотек та технологій для досягнення надійної і стабільної роботи, а також показує, як сучасні технології можуть бути використані для підвищення безпеки та комфорту в повсякденному житті.

3 ТЕСТУВАННЯ СИСТЕМИ

3.1 Тестування загальної працездатності системи

Тестування програмного забезпечення є важливим етапом розробки, який забезпечує виявлення та усунення помилок перед введенням системи в експлуатацію. У цьому розділі детально описується підхід до тестування коду, тестові сценарії, методи перевірки та результати тестування, які підтверджують надійність та правильність роботи розробленої системи. Метою тестування є перевірка коректної роботи всіх компонентів системи, включаючи датчики, мікроконтролер, інтерфейси з користувачем та систему сповіщень. Тестування дозволяє виявити можливі помилки у взаємодії компонентів, забезпечити стабільність системи під різними умовами експлуатації та підтвердити виконання заданих функцій. Для проведення тестування були підготовлені наступні компоненти:

1. Датчики, DHT11 для вимірювання температури та вологості, MQ2 для виявлення газу, датчик полум'я.

2. Мікроконтролер ESP32.

3. Платформа для віддаленого моніторингу та сповіщень - Blynk.

4. LCD-дисплей для локального відображення даних.

5. Середовище розробки - Arduino IDE.

Перед початком тестування було виконано налаштування мережевого з'єднання, ініціалізація датчиків та конфігурація платформи Blynk. Було розроблено ряд тестових сценаріїв для перевірки кожного аспекту роботи системи.

Ініціалізація компонентів. Ініціалізація всіх датчиків у функції setup(), після чого відбувається підключення до мережі Wi-Fi. Далі конфігурація платформи Blynk. На цьому етапі тестування необхідно переконатися, що всі датчики коректно ініціалізуються у функції setup(). Це включає в себе підключення всіх сенсорів та їх правильну конфігурацію. Наприклад, для

датчика DHT11, який вимірює температуру та вологість, необхідно викликати функцію dht.begin(), щоб підготувати його до роботи. Подібним чином для інших датчиків, таких як MQ2 (датчик газу) та датчик полум'я, необхідно встановити відповідні порти як вхідні або вихідні.

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

  // Ініціалізація датчика DHT11
  dht.begin();

  // Ініціалізація LCD-дисплея
  lcd.init();
  lcd.backlight();

  // Налаштування портів для датчиків газу та полум'я
  pinMode(MQ2_A, INPUT);
  pinMode(MQ2_D, INPUT);
  pinMode(FLAME_A, INPUT);
  pinMode(FLAME_D, INPUT);
  pinMode(BUZZER, OUTPUT);

  // Налаштування таймера для регулярного зчитування даних
  timer.setInterval(1000L, sendUptime);
}
```

3.2 Перевірка мережевого з'єднання

Другим кроком є підключення мікроконтролера до мережі Wi-Fi. Це необхідно для забезпечення передачі даних до платформи Blynk. Для цього в коді програми вказуються ім'я мережі (SSID) та пароль. Використовується функція Blynk.begin() для встановлення з'єднання з мережею Wi-Fi та сервером Blynk. Під час спроби підключення можна використовувати серійний монітор для відображення статусу з'єднання та діагностики можливих помилок.

```
#define BLYNK_TEMPLATE_ID "TMPL4DlyJdNk2"
#define BLYNK_TEMPLATE_NAME "TopTool"
#define BLYNK_AUTH_TOKEN "TBizLurLbq8zl2rvKGQ2fMbHTcIyvmEs"
#include <BlynkSimpleEsp32.h>

char auth[] = "TBizLurLbq8zl2rvKGQ2fMbHTcIyvmEs";
char ssid[] = "CybersecurityUSAID";
char pass[] = "";

void setup() {
  Serial.begin(115200);
  Serial.print("Connecting to WiFi...");
```

```
  // Підключення до Wi-Fi та Blynk
  Blynk.begin(auth, ssid, pass);

  while (Blynk.connected() == false) {
    Serial.print(".");
    delay(500);
  }
  Serial.println(" Connected!");

  // Подальша ініціалізація датчиків
  dht.begin();
  lcd.init();
  lcd.backlight();
  pinMode(MQ2_A, INPUT);
  pinMode(MQ2_D, INPUT);
  pinMode(FLAME_A, INPUT);
  pinMode(FLAME_D, INPUT);
  pinMode(BUZZER, OUTPUT);

  // Налаштування таймера
  timer.setInterval(1000L, sendUptime);
}
```

Конфігурація платформи Blynk включає в себе налаштування віртуальних пінів, до яких будуть надсилатися зчитані з датчиків дані. Це дозволяє моніторити показники в реальному часі через мобільний додаток або веб-інтерфейс Blynk. Функція `Blynk.virtualWrite()` використовується для відправки даних до відповідних віртуальних пінів на платформі Blynk.

Система успішно підключається до Wi-Fi мережі та передає дані до Blynk. Всі дані відображаються на платформі Blynk в реальному часі, що підтверджує коректну роботу функцій передачі даних та інтеграції з платформою Blynk.

3.3 Перевірка зчитуваних даних з датчиків

Зчитування даних з датчиків. Перевірка правильності зчитування даних з датчиків температури, вологості, газу, полум'я, напруги та струму. Спочатку потрібно ініціалізувати всі датчики у функції setup(). Потім іде функція sendUptime(), вона виконується періодично за допомогою таймера і відповідає за зчитування даних з усіх підключених датчиків. Наприклад, для зчитування даних з датчика DHT11 використовуються методи dht.readHumidity() та

dht.readTemperature(). Дані з інших датчиків зчитуються за допомогою функцій analogRead() та digitalRead().

```
voltage = pzem.voltage();
current = pzem.current();
W = pzem.power();
gas_value = analogRead(MQ2_A);
gas_status = digitalRead(MQ2_D);
humi = dht.readHumidity();
temp = dht.readTemperature();
flame_status = digitalRead(FLAME_D);
flame_value = analogRead(FLAME_A);
```

Далі ми перевіряли коректність роботи датчиків, перевіряли значення та звіряли їх з тими значеннями які відображалися на LCD-екрані. Результати, виявилися успішними, ініціалізація пройшла успішно, всі дані зчитуються коректно, також дані відображаються на LCD-дисплеї у реальному часі без затримок та помилок. Це підтверджує правильну роботу датчиків та їх інтеграцію з мікроконтролером ESP32.

3.4 Перевірка виявлення граничних умов

Виявлення граничних умов є критично важливим для забезпечення безпеки та надійності системи. Цей етап тестування перевіряє, чи система коректно реагує на перевищення граничних значень таких параметрів, як температура, газ і полум'я. Мета цієї перевірки - переконатися, що система здатна виявляти критичні ситуації та своєчасно реагувати на них, сповіщаючи користувача і активуючи відповідні механізми безпеки, такі як бузер.

Порядок проведення перевірки:

1. Імітація граничних температур;

2. Імітація присутності газу;

3. Імітація полум'я;

Імітація граничних температур. Висока температура - для перевірки реакції системи на високу температуру, ми імітували температуру, що перевищує 45°C. Оскільки створити такі умови в домашніх умовах складно, ми

знизили граничне значення до, наприклад, 20°C, щоб перевірити реакцію системи. Заміна фактичного порога дозволяє нам безпечно імітувати екстремальні умови.

Використовуючи нагрівач або інше джерело тепла, ми підвищували температуру поблизу датчика DHT11. Система повинна була виявити цю зміну і відправити відповідне сповіщення.

При досягненні граничного значення система відправляла сповіщення "temperature_is_high" і активувала бузер. Це підтвердило, що система коректно реагує на високу температуру.

Далі, потрібно перевірити низьку температуру. Для перевірки реакції на низьку температуру ми імітували умови нижче -5°C, знизивши поріг до, наприклад, 10°C для зручності тестування.

Використовуючи охолоджувальні елементи або розміщуючи датчик у холодильнику, ми знижували температуру поблизу датчика DHT11.

При зниженні температури до граничного значення система відправляла сповіщення "temperature_is_low". Це підтвердило здатність системи коректно реагувати на низьку температуру.

Наступною перевіркою, є перевірка реагування системи на газ. Для перевірки реакції на виявлення газу ми використовували запальничку, підносячи її до датчика MQ2 та випускаючи трохи газу.

Тестування проходило наступним чином, ми випускали газ в безпосередній близькості від датчика MQ2, що дало нам можливість перевірити роботу цього датчика.

В результаті, система виявляла присутність газу і відправляла сповіщення "gas_leaked". Це підтвердило здатність системи виявляти газ та своєчасно реагувати на його присутність.

Останньою перевіркою граничних умов є перевірка реагування на полум'я. Імітація присутності полум'я проводилася шляхом підпалювання

сірника і піднесення його до датчика полум'я, або запалюванням запальнички на відстані, як мінімум, 1 метр.

В результаті, система виявляла полум'я, відправляла сповіщення "flame" і активувала бузер. Це підтвердило, що система коректно реагує на виявлення полум'я.

Після проведених перевірок було виявлено, що система коректно реагує на всі задані граничні умови. При високій температурі система відправляла сповіщення "temperature_is_high" і активувала бузер, при низькій температурі - сповіщення "temperature_is_low", при виявленні газу - сповіщення "gas_leaked", а при виявленні полум'я - сповіщення "flame" і активацію бузера. Це підтверджує надійність та ефективність системи у виявленні та реагуванні на критичні ситуації.

3.5 Висновки по тестуванню

Проведене тестування системи продемонструвало високу надійність та ефективність її роботи в різних умовах експлуатації. У рамках тестування були перевірені основні функції системи, включаючи ініціалізацію компонентів, встановлення мережевого з'єднання, зчитування даних з датчиків та виявлення граничних умов. Результати тестування підтверджують, що система коректно функціонує відповідно до заданих вимог і здатна забезпечувати своєчасне сповіщення про небезпечні ситуації.

Основні результати тестування включають:

Тестування загальної працездатності системи. Успішно проведена ініціалізація всіх компонентів системи, включаючи датчики, мікроконтролер ESP32, LCD-дисплей та платформу для віддаленого моніторингу та сповіщень Blynk. Забезпечена коректна робота всіх сенсорів, що дозволяє системі точно вимірювати та передавати дані про температуру, вологість, наявність газу та

полум'я. Виконані тестові сценарії підтвердили стабільну роботу системи під різними умовами експлуатації, що дозволяє використовувати її для моніторингу критичних параметрів у реальному часі.

Перевірка мережевого з'єднання. Система успішно підключається до Wi-Fi мережі та платформи Blynk, що дозволяє віддалено моніторити всі показники через мобільний додаток або веб-інтерфейс. Дані з датчиків передаються до платформи Blynk без затримок, що підтверджує коректну інтеграцію мікроконтролера ESP32 з мережею та платформою Blynk.

Перевірка зчитуваних з датчиків даних: Коректність зчитування даних з датчиків підтверджена шляхом порівняння значень, отриманих з датчиків, з тими, що відображаються на LCD-дисплеї. Дані відображаються в реальному часі без помилок, що забезпечує точність вимірювань і надійність системи у збиранні даних.

Перевірка виявлення граничних умов. Система коректно реагує на високі та низькі температури, відправляючи відповідні сповіщення та активуючи бузер. Система виявляє присутність газу та полум'я, своєчасно сповіщаючи користувача про небезпеку. Імітація граничних умов показала, що система здатна надійно функціонувати навіть в екстремальних ситуаціях, що є критично важливим для забезпечення безпеки.

Результати тестування свідчать про те, що розроблена система відповідає всім заявленим вимогам і здатна забезпечувати надійний моніторинг критичних параметрів, таких як температура, вологість, наявність газу та полум'я. Система стабільно працює в різних умовах експлуатації, забезпечуючи своєчасне сповіщення користувача про небезпечні ситуації. Це підтверджує її готовність до введення в експлуатацію та використання в реальних умовах для забезпечення безпеки та моніторингу.

ВИСНОВКИ

У процесі виконання дипломної роботи була розроблена та впроваджена система моніторингу параметрів навколишнього середовища, яка використовує мікроконтролер ESP32 та сервіс Blynk для віддаленого управління та моніторингу даних. Система дозволяє зчитувати дані з датчиків температури, вологості, газу та полум'я, передавати ці дані на сервер Blynk і відображати їх у реальному часі на мобільному додатку, а також здійснювати сповіщення у разі перевищення допустимих значень.

Для збору даних були використані датчик температури та вологості DHT11, газовий датчик MQ-2, датчик полум'я, а також рідкокристалічний дисплей для локального відображення інформації. Використання цих датчиків дозволяє забезпечити надійний моніторинг ключових параметрів навколишнього середовища.

Програмне забезпечення було створено на основі бібліотек LiquidCrystal_I2C, DHT, WiFi, BlynkSimpleEsp32 та інших, що забезпечили взаємодію між апаратними компонентами та сервісом Blynk. Було досягнуто стабільної роботи системи з регулярним оновленням даних на сервері та локальному дисплеї.

Інтеграція з сервісом Blynk дозволила забезпечити віддалений доступ до даних через мобільний додаток, а також налаштувати систему сповіщень, що інформує користувача про перевищення допустимих рівнів температури, вологості, концентрації газу та наявності полум'я. Це значно підвищило функціональність та зручність використання системи.

Проведено детальний аналіз та оптимізацію кожного рядка коду для забезпечення ефективної роботи системи. В ході тестування були виявлені та виправлені можливі помилки, що дозволило досягти стабільної роботи системи в різних умовах.

Налаштування системи сповіщень в Blynk дозволило оперативно інформувати користувача про критичні зміни параметрів навколишнього середовища, такі як висока або низька температура, підвищений рівень газу та наявність полум'я. Це забезпечило високий рівень безпеки та контроль за ситуацією.

Також було проведено тестування приладу, датчиків, інтеграції з Blynk та роботи коду. Тестування показало, що прилад повністю готовий до використання у реальних умовах і може бути корисним у різних умовах використання, в серверній, в офісі чи навіть вдома.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційний сайт Arduino. URL: https://www.arduino.cc/ (дата звернення: 17.04.24)
2. Офіційний сайт Blynk. URL: https://blynk.io/ (дата звернення: 17.04.24)
3. Сайт Planetaklimata основнівимоги до температури в приміщеннях. URL: https://planetaklimata.com.ua/ua/news/?msg=2605 (дата звернення: 17.04.24)
4. Офіційний сайт PlatformIo. URL: https://platformio.org/ (дата звернення: 17.04.24)
5. Офіційний сайт Espressif. URL: https://www.espressif.com/en/products/sdks/esp-idf (дата звернення: 17.04.24)
6. Офіційний сайт Micropython. URL: https://micropython.org/ (дата звернення: 17.04.24)
7. Сайт Arduino, де розповідається про ESP32. URL: https://arduino.ua/ru/index.php?categoryID=122&show_all=yes (дата звернення: 17.04.24)
8. Офіційний сайт ArduinoIDE. URL: https://www.arduino.cc/en/software (дата звернення: 17.04.24)
9. Github репозиторій бібліотеки LiquidCrystal_I2C. URL: https://github.com/troublegum/liquidcrystal_i2c (дата звернення: 19.04.24)
10. Стаття про LiquidCrystal_I2C. URL: https://pdfs.semanticscholar.org/ba0d/e435da97caa776c44ed003cc0cfe4ad390e7.pdf (дата звернення: 19.04.24)
11. Github репозиторій бібліотеки Adafruit_LiquidCrystal. URL: https://github.com/adafruit/Adafruit_LiquidCrystal (дата звернення: 19.04.24)
12. Adafruit_LiquidCrystal. URL: https://www.arduino.cc/reference/en/libraries/adafruit-liquidcrystal/ (дата звернення: 19.04.24)

13. Github репозиторій бібліотеки NewLiquidCrystal. URL: https://github.com/fmalpartida/New-LiquidCrystal/blob/master/LiquidCrystal_I2C.h (дата звернення: 19.04.24)

14. DHT. URL: https://www.arduino.cc/reference/en/libraries/dht-sensor-library/ (дата звернення: 19.04.24)

15. Github репозиторій бібліотеки DHT. URL: https://github.com/adafruit/DHT-sensor-library (дата звернення: 19.04.24)

16. Github репозиторій бібліотеки Adafruit_Sensor. URL: https://github.com/adafruit/Adafruit_Sensor (дата звернення: 20.04.24)

17. Adafruit_Sensor. URL: https://electronoobs.com/eng_arduino_Adafruit_Sensor.php (дата звернення: 20.04.24)

18. Github репозиторій бібліотеки SimpleDHT. URL: https://github.com/winlinvip/SimpleDHT (дата звернення: 20.04.24)

19. SimpleDHT. URL: https://www.arduino.cc/reference/en/libraries/simpledht/ (дата звернення: 20.04.24)

20. Github репозиторій бібліотеки WiFi.h. URL: https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFi.h (дата звернення: 20.04.24)

21. Стаття про WiFi.h. URL: https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-9e9f6c84-5110-4032-89a5-04d346d1a699 (дата звернення: 20.04.24)

22. Github репозиторій бібліотеки ESP8266WiFi.h URL: https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/ESP8266 (дата звернення: 20.04.24)

23. ESP8266WiFi URL: https://qazf.com.ua/esp8266-wi-fi/ (дата звернення: 20.04.24)

24. Github репозиторій бібліотеки WiFiClient.h URL: https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFiClient.h (дата звернення: 20.04.24)

25. WiFiClient.h URL: https://doc.arduino.ua/ru/prog/WiFiClient (дата звернення: 20.04.24)

26. Github репозиторій бібліотеки AsyncTCP.h URL: https://github.com/me-no-dev/AsyncTCP/blob/master/src/AsyncTCP.h (дата звернення: 22.04.24)

27. AsyncTCP URL: https://www.arduino.cc/reference/en/libraries/asynctcp/ (дата звернення: 22.04.24)

28. Github репозиторій бібліотеки HardwareSerial.h URL: https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/HardwareSerial.h (дата звернення: 22.04.24)

29. Стаття про HardwareSerial URL: https://link.springer.com/chapter/10.1007/978-1-4842-9645-5_6 (дата звернення: 22.04.24)

30. Github репозиторій бібліотеки SoftwareSerial.h URL: https://github.com/arduino/ArduinoCore-avr/blob/master/libraries/SoftwareSerial/src/SoftwareSerial.h (дата звернення: 25.04.24)

31. SoftwareSerial.h URL: https://doc.arduino.ua/ru/prog/SoftwareSerial (дата звернення: 25.04.24)

32. Github репозиторій бібліотеки PZEM-004T-v30 URL: https://github.com/mandulaj/PZEM-004T-v30 (дата звернення: 25.04.24)

33. PZEM-004T-v30 URL: https://www.arduino.cc/reference/en/libraries/pzem004tv30/ (дата звернення: 25.04.24)

34. Github репозиторій бібліотеки EmonLib URL: https://github.com/openenergymonitor/EmonLib (дата звернення: 25.04.24)

35. EmonLib URL: https://www.arduino.cc/reference/en/libraries/emonlib/ (дата звернення: 25.04.24)

36. Github репозиторій бібліотеки BlynkSimpleEsp32.h URL: https://github.com/blynkkk/blynk-library/blob/master/src/BlynkSimpleEsp32.h (дата звернення: 27.04.24)

37. IoT – Інтернет речей URL: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82_%D1%80%D0%B5%D1%87%D0%B5%D0%B9 (дата звернення: 27.04.24)

38. Github репозиторій бібліотеки Firebase ESP32 URL: https://github.com/mobizt/Firebase-ESP32 (дата звернення: 29.04.24)

39. Сервіс Firebase URL: https://firebase.google.com/ (дата звернення: 29.04.24)

40. Сервіс Google Analytycs URL: https://marketingplatform.google.com/about/analytic (дата звернення: 30.04.24)

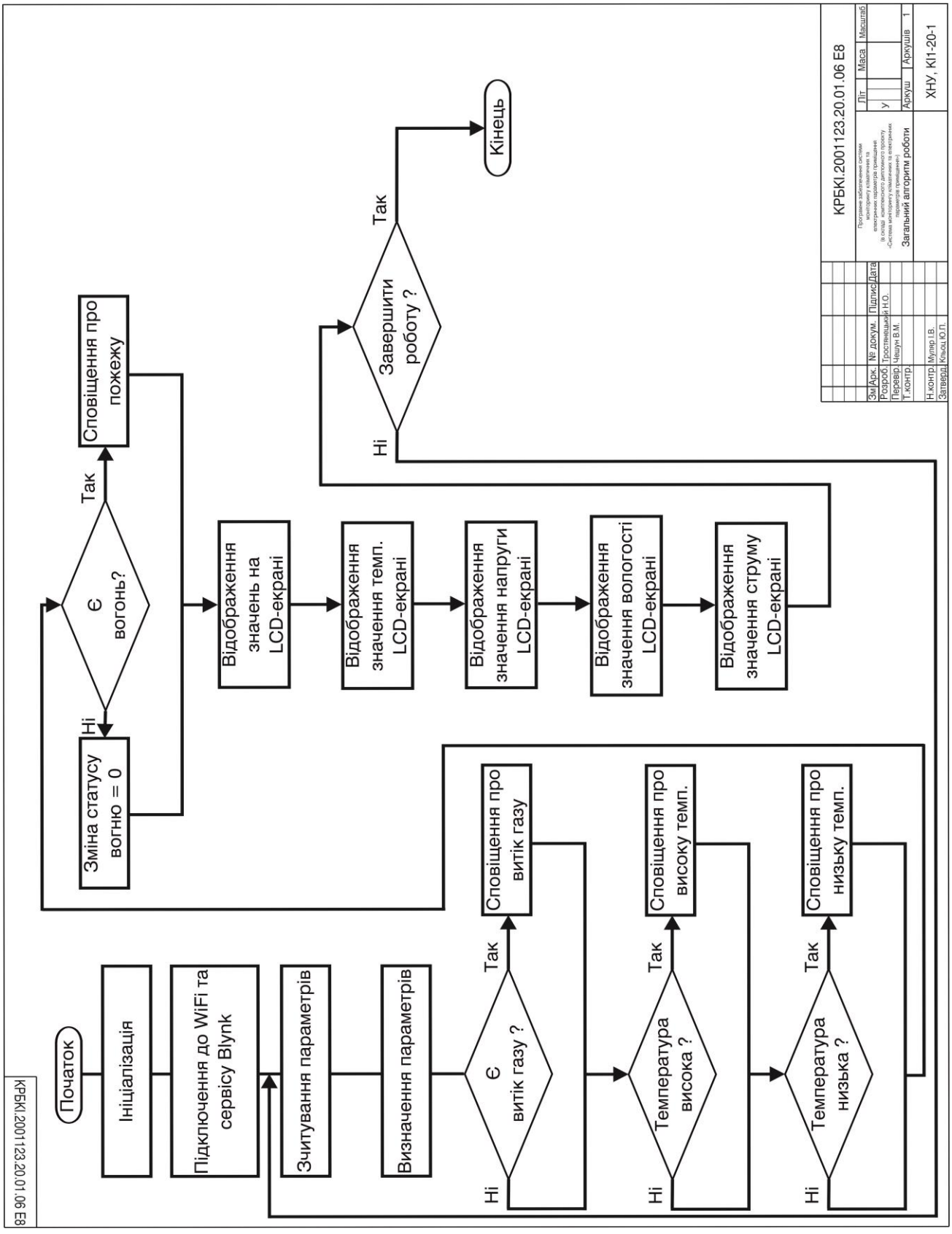
41. Сервіс Google Cloud URL: https://cloud.google.com/ (дата звернення: 30.04.24)

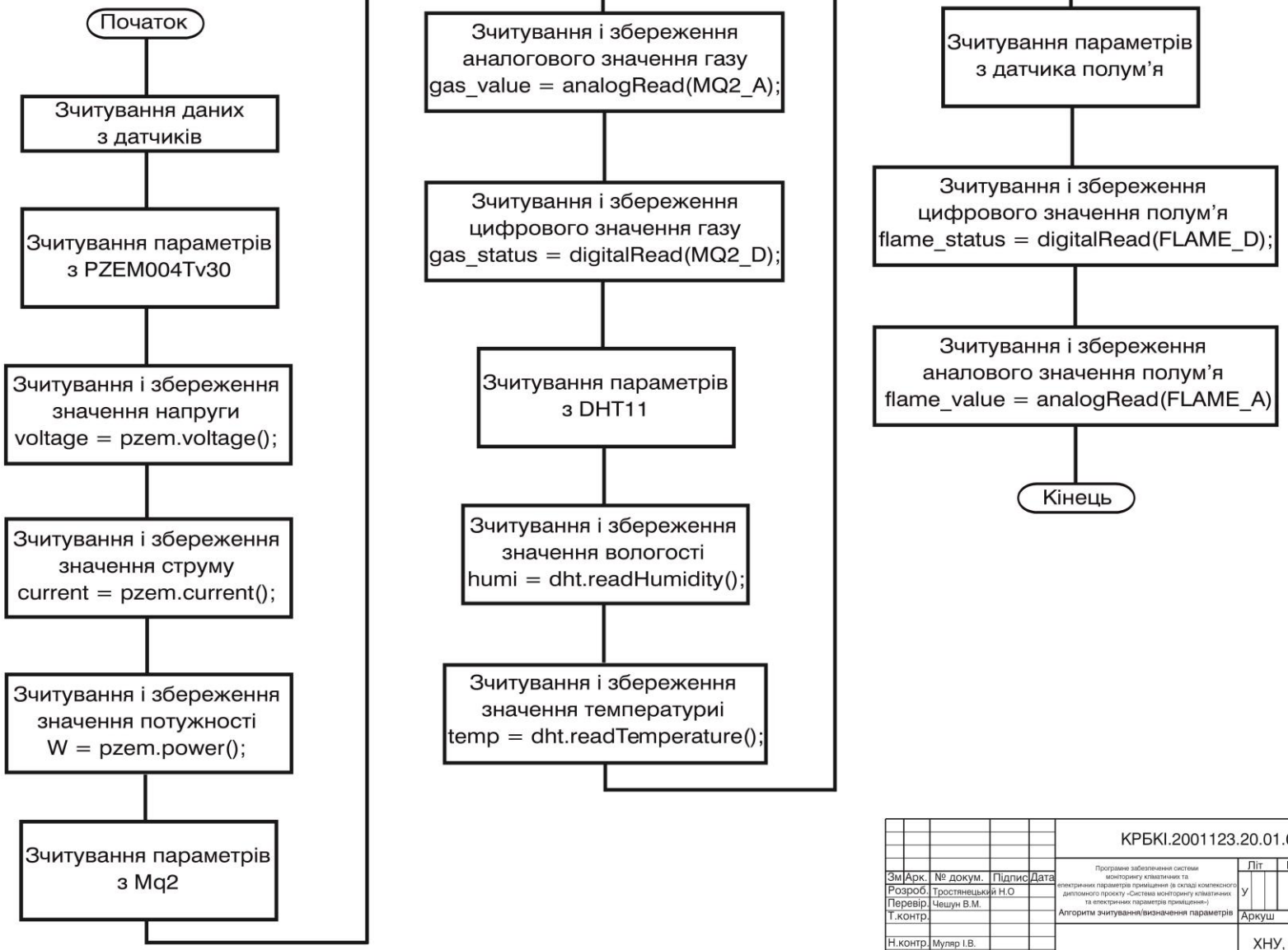
42. Rasberry Pi URL: https://www.raspberrypi.org/ (дата звернення: 01.05.24)

ДОДАТОК А

(обов'язковий)

Копія графічної частини





Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Тростянецький Н.О		
Перевір.		Чешун В.М.		
Т.контр.				
Н.контр.		Муляр І.В.		
Затверд.		Кльоц Ю.П.		

КРБКІ.2001123.20.01.06 Е8

Програмне забезпечення системи моніторингу кліматичних та електричних параметрів приміщення (в складі комплексного дипломного проекту «Система моніторингу кліматичних та електричних параметрів приміщення»)

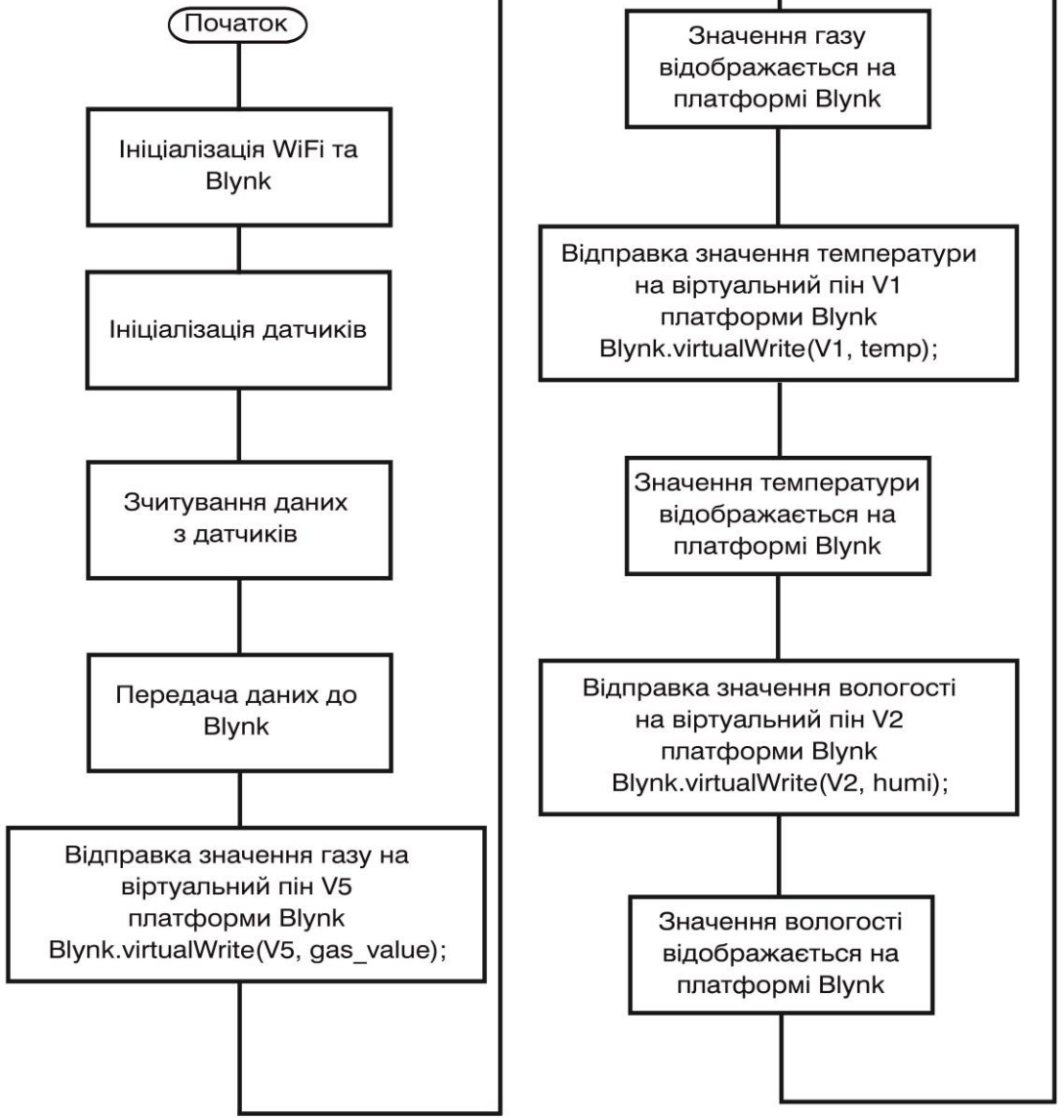
Алгоритм зчитування/визначення параметрів

Літ	Маса	Масштаб
У		

Аркуш | Аркушів 1

ХНУ, КІ1-20-1

Зм. Арк. № докум. Підпис Дата
КРБКІ. 2001123.20.01.06 ПЗ



					КРБКІ.2001123.20.01.06 Е8
Зм.	Арк.	№ докум.	Підпис	Дата	Програмне забезпечення системи моніторингу кліматичних та електричних параметрів приміщення (в складі комплексного дипломного проекту «Система моніторингу кліматичних та електричних параметрів приміщення»)
Розроб.		Тростянецький Н.О			Алгоритм відправки даних на Blynk
Перевір.		Чешун В.М.			
Т.контр.					
Н.контр.		Муляр І.В.			
Затверд.		Кльоц Ю.П.			

Літ	Маса	Масштаб
У		
Аркуш	Аркушів 1	
ХНУ, КІ1-20-1		

Зм. | Арк. | № докум. | Підпис | Дата

КРБКІ. 2001123.20.01.06 ПЗ

Арк. 83

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Тростянецького Назара Олексійовича
ПІБ здобувача вищої освіти

Студента ФІТ, 4 курсу, групи КІ1-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

11.06.24
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 131416 Назва: Програмне забезпечення системи моніторингу кліматичних та електричних параметрів приміщення (в складі комплексного дипломного проєкту "Система моніторингу кліматичних та електричних параметрів приміще... Додано в БД: 2024-06-18 Автора: Тростянецький Н.О. Керівники: Кльоц Ю.П. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	110194	953	662 (1%)	7 (1%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра кібербезпеки

ID перевірки:
1016373395

Дата перевірки:
18.06.2024 22:40:01 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
18.06.2024 23:03:11 EEST

ID користувача:
100008300

Назва документа: **Тростянецький_плагіат**

Кількість сторінок: **76** **Кількість слів:** **16489** **Кількість символів:** **123489** **Розмір файлу:** **2.61 MB** **ID файлу:** **1016180884**

1.89%
Схожість

Найбільша схожість: **0.58%** **з джерелом з Бібліотеки** (ID файлу: **1016163175**)

1.41% Джерела з Інтернету	283	Сторінка 78
0.96% Джерела з Бібліотеки	101	Сторінка 79

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Програмне забезпечення системи моніторингу кліматичних умов та електричних параметрів приміщення (в складі комплексного дипломного проєкту "система моніторингу кліматичних та електричних параметрів приміщення").

Автор: Тростянецький Назар Олексійович

Спеціальність: 123 – Компютерна інженерія

Освітня програма: Прграмуваңня та захист комп'ютерних систем і мереж

Науковий керівник: Кльоц Юрій Павлович, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою Unichek складає 98,11%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism v-15.257 складає 99%.

Згідно з Положенням про систему забезпечення академічної доброчесності у ХНУ (https://khmnu.edu.ua/wp-content/uploads/normatyvni-dokumenty/polozhennya/pro-systemu-zabezpechennya-akademichnoyi-dobrochesnosti.pdf, Додаток В) кваліфікаційна робота, виконана за , освітньо-професійною програмою, кількісні показники рівня унікальності тексту у відсотках до загального обсягу матеріалу в якій складає 75-100 %, визнається роботою з високою унікальністю тексту: «Текст вважається унікальним і не потребує додаткових дій щодо запобігання неправомірним запозиченням».

Керівник роботи _____ Юрій КЛЬОЦ

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Тростянецький Назар Олексійович

Тема Програмне забезпечення системи моніторингу кліматичних умов та електричних параметрів приміщення (в складі комплексного дипломного проєкту "система моніторингу кліматичних та електричних параметрів приміщення")

Спеціальність 123 – Комп'ютерна інженерія

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 80.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі було розроблене програмне забезпечення система моніторингу кліматичних умов та електричних параментрів приміщення

2. Висновок про відповідність кваліфікаційної роботи завданню У кваліфікаційній роботі було виконано поставлене завдання як у теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі роботи наведена загальна характеристика задачі, визначені об'єкт, предмет та методи дослідження, а також сформульована мета. Зазначені задачі, що потрібно виконати для досягнення поставленої мети, проведений аналіз досліджуваної проблеми та обгрунтований підхід до її вирішення. У першому розділі розглядається середовища розробки та бібліотеки. Наступні розділи присвячені розробці програмного забезпечення, реагуванню на граничні значення і налаштуванню зв'язку з сервісом Blynk. Також було проведено тестування системи.

4. Позитивні сторони роботи Кваліфікаційна робота має практичну цінність. Вона полягає у розробці програмного забезпечення для приладу, який буде встановлено в серверній кімнаті, для моніторингу температури, вологості, струму та напруги. Завдяки цьому пз, прилад зможе функціонувати належним чином і реагувати на певні події

5. Негативні сторони роботи В системі немає негативних сторін.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. В цілому, графічне оформлення є якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень для досягнення мети.

8. Інші зауваження В переліку використаних джерел наявні посилання на популярні ресурси, такі, як Вікіпендія, які не рекомендовано використовувати при написанні кваліфікаціних робіт.

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Бойко Юрій Миколайович,
професор кафедри ТМІТ, доктор технічних наук

« 16 » червня 2024. (підпис)