МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
Циклова комісія *комп'ютерних систем та мереж*
(повна назва циклової комісії)

Допустити до захисту
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
_____ *Ірина КРАВЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)
« 10 » 06 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-ПРОФЕСІЙНОГО СТУПЕНЯ
ФАХОВИЙ МОЛОДШИЙ БАКАЛАВР

Тема: *Проектування фізичної моделі карти повітряних тривог України на базі мікроконтролера ESP8266.*

Група: *3-013* Спеціальність: *123 «Комп'ютерна інженерія»*

Здобувач освіти _____ *Іван САГАЙДАК*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ *Роман МІНЕНКО*
(підпис) (ім'я, ПРІЗВИЩЕ)

Консультант з оформлення пояснювальної записки _____ *Оксана ОСАДЧА*
(підпис) (ім'я, ПРІЗВИЩЕ)

Кривий Ріг 2025 р.

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Відділення *комп'ютерної та програмної інженерії*
Циклова комісія *комп'ютерних систем та мереж*
Освітньо-професійний ступінь *фаховий молодший бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
(підпис) *Ірина КРАВЧУК*
(ім'я, ПРІЗВИЩЕ)
« 10 » 05 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

САГАЙДАК Іван Олександрович
(прізвище, ім'я, по батькові)

1.Тема роботи *Проектування фізичної моделі карти повітряних тривог України на базі мікроконтролера ESP8266.*

Керівник роботи *Міненко Роман Вадимович, к. ф.-м. н.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по коледжу від « *04* » *04* 2025 року № *50-ст*

2. Строк подання здобувачем освіти роботи з 19.05.2025 по *13.06.2025*

3. Вихідні дані до роботи
Середовище розробки Arduino IDE для прошивки мікроконтролера ESP8266, навчальна та методична література, наукові джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Предметна область та проєктування системи
Розробка апаратно-програмного комплексу
Тестування та демонстрація роботи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Презентація Microsoft PowerPoint

6. Консультанти розділів роботи (проекту)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Узгодження технічного завдання з керівником кваліфікаційної роботи*	*27.02.2025-04.03.2025*	*виконано*
2	*Підбір та вивчення науково-технічної літератури за темою кваліфікаційної роботи*	*14.03.2025-29.03.2025*	*виконано*
3	*Обґрунтування вибору програмних засобів*	*18.04.2025-20.04.2025*	*виконано*
4	*Опис компонентів. Обґрунтування їх вибору.*	*21.04.2025-22.04.2025*	*виконано*
5	*Розробка програмного забезпечення*	*23.04.2025-27.04.2025*	*виконано*
6	*Дослідження ефективності реалізованих методів.*	*29.04.2025-02.05.2025*	*виконано*
7	*Написання пояснювальної записки*	*12.05.2025-23.05.2025*	*виконано*
8	*Перевірка на плагіат пояснювальної записки*	*26.05.2025-30.05.2025*	*виконано*
9	*Попередній захист кваліфікаційної роботи*	*02.06.2025-06.06.2025*	*виконано*
10	*Захист кваліфікаційної роботи*		

Здобувач освіти _____ *Іван САГАЙДАК*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ *Роман МІНЕНКО*
(підпис) (ім'я, ПРІЗВИЩЕ)

StrikePlagiarism.com

Дата звіту 6/9/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Ukrainian national aviation university
Заголовок
Сагайдак_Іван_Олександрович_3013_2025_123
Автор Науковий керівник / Експерт
СагайдакМіненко Р
підрозділ
Криворізький Фаховий коледж

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

3.79%
3.79%
КП 1

2.15%
2.15%
КЦ

25
Довжина фрази для коефіцієнта подібності 2

11490
Кількість слів

93363
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		2
Інтервали	A→	0
Мікропробіли	·	76
Білі знаки		0
Парафрази (SmartMarks)	a	28

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	http://kk.nau.edu.ua/article/4143	87 0.76 %
2	Створення додатку для візуалізації лекцій з системного програмування [redacted] 6/4/2025 International University of Economics and Humanities named after Academician Stepan Demianchuk (International University of Economics and Humanities named after Academician Stepan Demianchuk)	21 0.18 %
3	Бібліотеки України в цифровому медіапросторі: теоретико-методологічні засади розвитку Мар'їна Олена Юріївна	20 0.17 %

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Проектування фізичної моделі карти повітряних тривог України на базі мікроконтролера *ESP*8266» містить: *53* сторінки основного тексту, *18* рисунків, *17* використаних джерел, *1* додаток.

ESP8266, WS2812B, OLED SSD1306, TTP223, API, IOT, ПРОКСІ-СЕРВІС, FASTAPI, PYTHON, ARDUINO IDE, JSON, HTTP, WI-FI, MDNS, EEPROM, OTA, ПОВІТРЯНА ТРИВОГА, ВІЗУАЛІЗАЦІЯ.

У даній кваліфікаційній роботі розробляється апаратно-програмний комплекс для візуалізації актуального стану повітряних тривог на фізичній карті України. Система складається з пристрою на базі мікроконтролера *ESP*8266, що керує адресною світлодіодною стрічкою та *OLED*-дисплеєм, та проміжного проксі-сервісу для обробки даних.

Метою роботи є створення надійного та інтуїтивно зрозумілого засобу інформування населення про повітряні загрози, який доповнює існуючі цифрові канали сповіщення. Актуальність роботи зумовлена критичною важливістю оперативного та доступного інформування громадян в умовах воєнного стану, а також недоліками існуючих рішень, такими як залежність від мобільних пристроїв та стабільності мереж.

Основними результатами роботи є: розроблена фізична модель карти України з *LED*-індикацією для 29 регіонів; програмне забезпечення для мікроконтролера *ESP*8266, що забезпечує отримання даних про тривоги, керування індикацією, відображення службової інформації, конфігурування через веб-інтерфейс та *OTA*-оновлення; проксі-сервіс на *FastAPI* для агрегації та перетворення даних з офіційного *API* (*alerts.in.ua*) у компактний формат для *ESP*8266, що підтримує різні типи тривог та зональну систему.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API (англ. *Application Programming Interface*) – програмний інтерфейс застосунку, набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

AP (англ. *Access Point*) – точка доступу, режим роботи *Wi-Fi* модуля, при якому він створює власну бездротову мережу.

Bearer Token – тип токену доступу, що використовується в *HTTP*-аутентифікації для надання доступу до захищених ресурсів.

EEPROM (англ. *Electrically Erasable Programmable Read-Only Memory*) – електрично стираюча програмована постійна пам'ять, тип незалежної пам'яті, що використовується в мікроконтролерах для зберігання налаштувань.

***ESP*8266** – мікроконтролер з вбудованим *Wi-Fi* модулем, популярний для проектів Інтернету речей.

HTML (англ. *HyperText Markup Language*) – мова гіпертекстової розмітки, стандартна мова для створення веб-сторінок.

I2C (англ. *Inter-Integrated Circuit*) – послідовна шина даних для зв'язку інтегральних схем, що використовується, наприклад, для підключення *OLED*-дисплеїв.

IoT (англ. *Internet of Things*) – Інтернет речей, концепція обчислювальної мережі фізичних предметів («речей»), оснащених вбудованими технологіями для взаємодії один з одним або із зовнішнім середовищем.

JSON (англ. *JavaScript Object Notation*) – текстовий формат обміну даними, заснований на *JavaScript*, що легко читається людьми та обробляється комп’ютерами.

LED (англ. *Light-Emitting Diode*) – світлодіод, напівпровідниковий пристрій, що випромінює світло при пропусканні через нього електричного струму.

mDNS (англ. *Multicast DNS*) – протокол, що дозволяє розпізнавати імена хостів у локальній мережі без необхідності налаштування спеціального *DNS*-сервера.

NTP (англ. *Network Time Protocol*) – мережевий протокол для синхронізації годинників комп'ютерних систем через мережі з комутацією пакетів.

OLED (англ. *Organic Light-Emitting Diode*) – органічний світлодіод, технологія дисплеїв, що забезпечує високу контрастність та низьке енергоспоживання.

OTA (англ. *Over-The-Air*) *Update* – оновлення програмного забезпечення "по повітрю", без фізичного підключення до пристрою.

SSID (англ. *Service Set Identifier*) – ідентифікатор набору служб, унікальне ім'я бездротової мережі *Wi-Fi*.

Wi-Fi (англ. *Wireless Fidelity*) – технологія бездротового зв'язку, що дозволяє електронним пристроям підключатися до комп'ютерної мережі.

ВСТУП

Традиційні системи оповіщення, такі як сирени, мають обмеження щодо покриття та ефективності в умовах щільної міської забудови. Існуючі цифрові рішення, незважаючи на свою популярність серед користувачів мобільних додатків та веб-сервісів, значною мірою залежать від стабільності інтернет-зв'язку, мобільних мереж, рівня заряду пристроїв та активної уваги користувача.

Зазначені обмеження створюють потребу в розробці альтернативних, надійних та інтуїтивно зрозумілих засобів візуалізації стану повітряних тривог, які могли б функціонувати більш автономно та забезпечувати швидке сприйняття інформації. Фізична модель карти повітряних тривог, що відображає актуальну ситуацію за допомогою світлової індикації, може стати ефективним доповненням до існуючих систем. Такий пристрій забезпечує постійну інформацію про безпекову обстановку, не вимагаючи від користувача активних дій для її отримання. Кольорове кодування регіонів дозволяє миттєво оцінити масштаб загрози, що є особливо важливим у стресових ситуаціях.

Дана кваліфікаційна робота безпосередньо пов'язана з актуальними завданнями підвищення рівня інформованості та безпеки громадян України. Розробка спрямована на практичне застосування технологій інтернету речей для вирішення соціально значущої проблеми та відповідає сучасним тенденціям розвитку систем оперативного оповіщення, маючи потенціал для інтеграції у ширші концепції «розумного міста» або систем цивільного захисту.

Метою даної кваліфікаційної роботи є розробка апаратно-програмного комплексу для візуалізації актуального стану повітряних тривог на фізичній карті України, що забезпечує надійне та інтуїтивно зрозуміле інформування користувачів.

Програмна частина роботи передбачає створення прошивки для мікроконтролера *ESP*8266 з підтримкою підключення до *Wi-Fi* мережі, отримання даних про тривоги з *API*, керування світлодіодною індикацією для 29 регіонів, відображення службової інформації на *OLED*-дисплеї, обробки взаємодії з

користувачем через сенсорну кнопку, надання веб-інтерфейсу для налаштування параметрів системи, підтримки *OTA*-оновлення та збереження налаштувань в енергонезалежній пам'яті.

Предметом дослідження є апаратно-програмний комплекс для візуалізації карти повітряних тривог України, що включає фізичну модель карти з інтегрованими світлодіодними індикаторами, мікроконтролер *ESP*8266 та пов'язані з ним апаратні компоненти, програмне забезпечення для мікроконтролера, проміжний проксі-сервіс на *Python*, методи отримання та обробки даних про повітряні тривоги, а також інтерфейси взаємодії користувача з системою.

РОЗДІЛ 1
ПРЕДМЕТНА ОБЛАСТЬ ТА ПРОЄКТУВАННЯ СИСТЕМИ

1.1 Аналіз задачі візуалізації повітряних тривог

В умовах воєнного стану інформування населення про повітряні тривоги стало критично важливим завданням національної безпеки. Система повітряного оповіщення має забезпечувати швидке та надійне доведення інформації про загрозу до кожного громадянина, незалежно від його місцезнаходження та технічних можливостей.

Традиційні засоби оповіщення, такі як сирени повітряної тривоги, мають обмежений радіус дії та не завжди ефективні в умовах сучасної забудови міст. Електронні засоби комунікації, включаючи мобільні додатки та *SMS*-повідомлення, залежать від стабільності мережі зв'язку, яка може бути порушена під час військових дій. У цьому контексті виникає потреба в альтернативних способах візуалізації стану повітряних тривог, які були б автономними, надійними та інтуїтивно зрозумілими.

Задача візуалізації повітряних тривог включає в себе декілька ключових аспектів. По-перше, необхідно забезпечити актуальність інформації про стан тривоги в режимі реального часу. Це вимагає інтеграції з офіційними джерелами даних та мінімізації затримок у передачі інформації. По-друге, візуалізація має бути регіональною, оскільки повітряні тривоги оголошуються для конкретних областей України, а не для всієї території одночасно. По-третє, система повинна чітко розрізняти різні типи сигналів: початок тривоги, її закінчення, а також можливі попередження або часткові скасування.

Географічний аспект візуалізації є особливо важливим для розуміння масштабу загрози. Карта України з підсвічуванням областей під тривогою дозволяє користувачам швидко оцінити ситуацію не лише у власному регіоні, але й отримати загальне уявлення про безпекову обстановку в країні. Це особливо актуально для людей, які мають родичів або інтереси в різних регіонах, а також для координаторів гуманітарних організацій та служб порятунку.

1.2 Огляд існуючих рішень та їх недоліки

Сучасний ринок пропонує широкий спектр рішень для інформування про повітряні тривоги, кожне з яких має свої особливості та обмеження. Найпоширенішими є мобільні додатки, веб-сервіси, системи *push*-сповіщень та традиційні засоби оповіщення.

Мобільні додатки, такі як «Повітряна тривога», стали основним джерелом інформації для більшості українців. Ці додатки надають детальну інформацію про стан тривог у режимі реального часу, включають карти з візуалізацією та можливість налаштування сповіщень для конкретних регіонів. Однак вони мають суттєві недоліки. По-перше, залежність від заряду мобільного пристрою та стабільності мобільної мережі. Під час масованих атак оператори зв'язку можуть відчувати перевантаження, що призводить до затримок у доставці сповіщень. По-друге, користувач має активно відкривати додаток або перевіряти сповіщення, що може бути пропущено під час сну або зосередженої роботи.

Веб-сервіси та інтернет-портали пропонують більш детальну аналітику та статистику щодо повітряних тривог, але вони потребують постійного доступу до комп'ютера та інтернету. Такі ресурси зазвичай не надають миттєвих сповіщень і вимагають від користувача регулярного оновлення сторінки для отримання актуальної інформації. Крім того, в умовах стресових ситуацій навігація по веб-сайту може бути складною та відволікаючою.

Системи автоматичних *SMS*-сповіщень від мобільних операторів та державних служб мають високу надійність доставки, але вони обмежені в можливостях візуалізації. Текстове повідомлення не може передати масштаб загрози або географічний контекст тривоги. Також *SMS*-сповіщення можуть мати затримки, особливо під час пікових навантажень на мережу.

Традиційні сирени повітряної тривоги залишаються важливим елементом системи оповіщення, але їх ефективність у сучасних умовах знижується. Звукові сигнали можуть бути не почуті в приміщеннях з хорошою звукоізоляцією, а також їх важко розрізнити серед інших звуків міського середовища. Сирени також не

передають інформацію про те, в якому саме регіоні оголошена тривога, якщо людина знаходиться близько до кордону областей.

Телевізійні та радіопередачі включають регулярні оновлення про стан повітряних тривог, але вони вимагають від користувача активного слухання або перегляду. Інформація передається через певні інтервали часу, що може призвести до затримок у сприйнятті критично важливих змін у ситуації.

1.3 Визначення функціональних вимог до системи

Вимоги становлять фундаментальну основу для архітектурного проєктування, обґрунтування технологічних рішень та розробки як апаратних, так і програмних компонентів системи. Вони визначають ключові функціональні можливості, які система повинна забезпечувати, та описують специфіку її поведінки за різних експлуатаційних умов.

Центральною функціональною вимогою є забезпечення точного та своєчасного відображення актуального стану повітряних тривог для заданого переліку адміністративно-територіальних одиниць України. Фізична модель карти має забезпечувати чітку та однозначну індикацію присутності або відсутності тривоги в кожному регіоні, який моніториться. Система повинна підтримувати розпізнавання основного типу загрози – повітряної тривоги, включаючи артилерійський обстріл, загрозу вуличних боїв, хімічну або ядерну небезпеку. Кожен регіон карти обладнується індивідуальним світловим індикатором, параметри якого – колір, інтенсивність освітлення, режим миготіння – відповідають поточному статусу тривоги. Критично важливим є мінімізація затримки між появою нових даних у джерелі та їх відображенням на карті.

Для реалізації основної функції візуалізації система повинна забезпечувати стабільне та надійне отримання інформації про повітряні тривоги з попередньо визначеного зовнішнього джерела даних. Це передбачає здатність підключення до спеціалізованого програмного інтерфейсу, який агрегує та транслює актуальну інформацію про тривоги.

Наступною критичною функціональною вимогою є ефективна обробка та інтерпретація отриманих даних. Система повинна здійснювати трансформацію сирих даних, отриманих від *API*, у формат, безпосередньо придатний для керування індикаторами на фізичній моделі. Це включає парсинг структури даних, ідентифікацію регіонів та відповідних їм статусів тривог. Необхідно реалізувати логіку зіставлення кодів або назв регіонів з *API* на конкретні фізичні світлодіоди карти. У випадках одночасного надходження інформації про кілька типів тривог для одного регіону, система має використовувати заздалегідь визначений алгоритм пріоритезації для відображення найбільш критичної загрози.

Система повинна надавати користувачеві можливості конфігурування основних параметрів роботи. Налаштування підключення до бездротової мережі включає введення та збереження ідентифікатора мережі та пароля для доступу до локальної *Wi-Fi* мережі, що є обов'язковою умовою для отримання даних з мережі. Конфігурація параметрів доступу до *API* передбачає можливість введення та збереження *URL*-адреси *API*-сервера, з якого завантажуватимуться дані про тривоги, а також відповідного токена авторизації.

Для забезпечення зручного доступу до налаштувань та моніторингу стану пристрою система повинна надавати веб-інтерфейс, доступний через локальну мережу. Цей інтерфейс має бути інтуїтивно зрозумілим та дозволяти користувачеві здійснювати початкове налаштування параметрів *Wi-Fi* та *API*, переглядати та модифікувати раніше збережені конфігурації, отримувати інформацію про поточний стан пристрою, включаючи статус підключення до *Wi-Fi*, поточну *IP*-адресу, час останнього успішного оновлення даних про тривоги. Додатково інтерфейс повинен дозволяти ініціювати системні операції, такі як перезавантаження мікроконтролера або скидання всіх налаштувань до заводських значень, а також здійснювати оновлення програмного забезпечення по повітрю, що значно спрощує процес підтримки та вдосконалення пристрою без необхідності фізичного підключення до комп'ютера.

У випадку використання в архітектурі системи проксі-сервісу для агрегації та попередньої обробки даних перед їх передачею на мікроконтролер, до цього

компонента висуваються специфічні функціональні вимоги. Проксі-сервіс повинен забезпечувати взаємодію з основним, більш складним *API* джерела даних про тривоги, обробляти отримані дані, фільтрувати їх та трансформувати у значно спрощений та компактний формат, оптимізований для пристроїв з обмеженими обчислювальними ресурсами. Типовим прикладом такого формату є рядок символів, де кожен символ кодує стан тривоги для конкретного регіону. Проксі-сервіс має надавати власний спрощений *API*-ендпоінт для *ESP*8266, через який мікроконтролер отримуватиме вже оброблені дані та забезпечувати механізм автентифікації для обмеження несанкціонованого доступу до свого *API*.

1.4 Обґрунтування технологічних рішень

1.4.1 Вибір мікроконтролера *ESP*8266 та апаратних компонентів

Серцем апаратної платформи системи виступає мікроконтролер, що відповідає за встановлення мережевого з'єднання, отримання та первинну обробку даних про повітряні тривоги, а також управління всіма периферійними пристроями візуалізації. Після ретельного порівняльного аналізу характеристик різних мікроконтролерних платформ, включаючи *Arduino Uno*, *Raspberry Pi*, *STM*32 та *ESP*32, було прийнято рішення використовувати мікроконтролер *ESP*8266. *NodeMCU esp8266* показано на рисунку 1.1.

Рисунок 1.1 – *NodeMCU esp8266*

Вибір *ESP*8266 обґрунтовується комплексом переваг, що роблять його оптимальним для поставленого завдання. Передусім, цей мікроконтролер

характеризується виключною економічною привабливістю, що є критично важливим фактором для створення доступного пристрою масового використання. Вартість *ESP*8266 в кілька разів нижча за альтернативні рішення з аналогічним функціоналом, що дозволяє мінімізувати собівартість кінцевого продукту без компромісів у функціональності.

Інтегрований *Wi-Fi* модуль *ESP*8266 забезпечує безпосереднє підключення до мережі Інтернет без необхідності використання зовнішніх мережевих адаптерів, що значно спрощує схемотехнічне рішення та зменшує загальні габарити пристрою. Підтримка стандартів *IEEE* 802.11 *b*/*g*/*n* на частоті 2.4 ГГц забезпечує стабільне з'єднання з більшістю домашніх та офісних мереж *Wi-Fi*. Вбудований *TCP*/*IP* стек дозволяє легко реалізувати *HTTP*-клієнт для взаємодії з веб-сервісами та *HTTP*-сервер для налаштування пристрою. *Esp*8266 показано на рисунку 1.2.

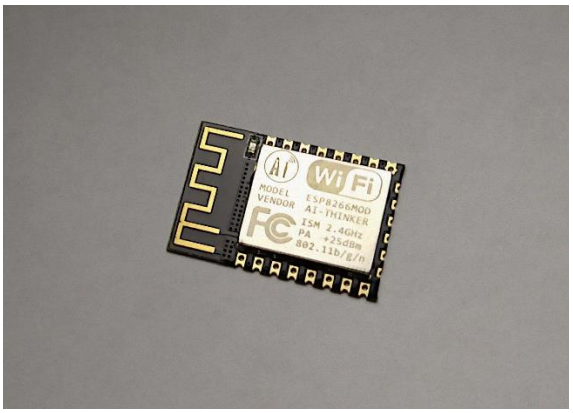

Рисунок 1.2 – *esp8266*

Обчислювальна потужність *ESP*8266 з 32-бітним процесором *Tensilica L*106, що працює на частоті 80 МГц (з можливістю розгону до 160 МГц), цілком достатня для виконання всіх поставлених завдань. Це включає декодування *JSON*-відповідей від проксі-сервісу, обробку 29-символьного рядка стану тривог, управління адресною світлодіодною стрічкою з індивідуальним контролем кольору кожного елемента, відображення інформації на *OLED*-дисплеї та обслуговування веб-інтерфейсу для користувацьких налаштувань. Обсяг оперативної пам'яті 80 КБ та флеш-пам'яті до 4 МБ забезпечує достатній простір для зберігання прошивки, веб-інтерфейсу та користувацьких конфігурацій.

Екосистема розробки для *ESP*8266 відзначається виключною зрілістю та підтримкою. Повна сумісність з популярним середовищем *Arduino IDE* значно спрощує процес розробки, налагодження та прошивання пристрою. Величезна спільнота розробників створила обширну базу бібліотек, прикладів коду та технічної документації, що прискорює процес розробки та дозволяє швидко вирішувати технічні питання. Наявність готових бібліотек для роботи з *Wi-Fi*, *HTTP*-клієнтом, веб-сервером, *JSON*-парсером та різноманітними периферійними пристроями мінімізує обсяг коду, що потребує написання з нуля.

Для реалізації візуального представлення стану повітряних тривог на фізичній карті України було обрано адресну світлодіодну стрічку типу *WS*2812*B*. Цей вибір зумовлений унікальними характеристиками даного типу світлодіодів, що ідеально відповідають вимогам проєкту. Кожен світлодіод *WS*2812*B* містить вбудований контролер, що дозволяє індивідуально керувати яскравістю та кольором кожного елемента стрічки, що є принципово важливим для точного відображення різних типів тривог у кожному з 29 регіонів України.

Технічні характеристики *WS*2812*B* включають підтримку 24-бітного кольору (16.7 мільйонів відтінків), що забезпечує широкі можливості для кольорового кодування різних типів повітряних тривог. *WS*2812*B* показано на рисунку 1.3. Високі показники яскравості (до 1000 мКд при струмі 20 мА на колір) гарантують добру видимість індикації навіть при денному освітленні. Однопровідний протокол управління дозволяє керувати всією стрічкою, використовуючи лише один цифровий пін мікроконтролера, що економить ресурси *GPIO* та спрощує монтаж.

Рисунок 1.3 – *WS2812B*

Протокол передачі даних *WS*2812*B* базується на часовій модуляції сигналу, де логічний нуль та одиниця кодуються різною тривалістю високого та низького рівнів сигналу. Це дозволяє передавати 24 біти інформації про колір (8 біт на кожен канал *RGB*) для кожного світлодіода послідовно по одному проводу. Швидкість передачі даних становить близько 800 кГц, що забезпечує достатню швидкодію для оновлення всієї стрічки без помітних затримок.

Для надання користувачеві додаткової інформації про поточний стан системи, час останнього оновлення даних, мережеві параметри або діагностичні повідомлення, в систему інтегровано монохромний *OLED*-дисплей на базі контролера *SSD*1306. Цей тип дисплеїв характеризується надзвичайно низьким енергоспоживанням (менше 20 мА в активному режимі), що важливо для загальної енергоефективності пристрою. Висока яскравість та контрастність *OLED*-технології забезпечують відмінну читабельність тексту в різних умовах освітлення. *OLED* 0.96 *I*2*C* показано на рисунку 1.4.

Рисунок 1.4 – *OLED* 0.96 *I*2*C*

Роздільна здатність 128*x*64 пікселі дозволяє комфортно відображати текстову інформацію різними шрифтами, а також прості графічні елементи, такі як індикатори рівня сигналу *Wi-Fi* або стан підключення до мережі. Підключення дисплея до *ESP*8266 здійснюється через інтерфейс *I*2*C*, що потребує лише два піни мікроконтролера (*SDA* та *SCL*) та дозволяє легко розширювати систему додатковими *I*2*C*-пристроями в майбутньому.

Контролер *SSD*1306 підтримує різноманітні режими відображення, включаючи прокрутку тексту, інверсію зображення та регулювання яскравості. Вбудований графічний *RAM* обсягом 1 КБ дозволяє зберігати повний кадр зображення, що забезпечує плавне оновлення контенту без мерехтіння.

Для забезпечення інтуїтивної взаємодії з користувачем, зокрема для перемикання режимів відображення інформації на *OLED*-дисплеї або активації спеціальних функцій, таких як скидання мережевих налаштувань, в систему інтегровано ємнісну сенсорну кнопку на базі мікросхеми *TTP*223. Цей вибір зумовлений кількома практичними перевагами над традиційними механічними кнопками.

Ємнісні сенсорні кнопки не мають рухомих частин, що забезпечує їхню високу надійність та довговічність. Відсутність механічного зносу гарантує стабільну роботу протягом мільйонів операцій спрацювання. Герметична конструкція робить їх нечутливими до пилу, вологи та інших забруднень, що важливо для пристрою, який може експлуатуватися в різних умовах.

*TTP*223 забезпечує стабільне спрацювання при дотику людини, автоматично калібруючи чутливість залежно від умов навколишнього середовища. Вбудований фільтр перешкод мінімізує хибні спрацювування від електромагнітних наводок. Вихідний сигнал представляє собою стандартний цифровий рівень, сумісний з *GPIO*-пінами *ESP*8266 без необхідності додаткових узгоджувальних схем. *TTP*223 показано на рисунку 1.5.

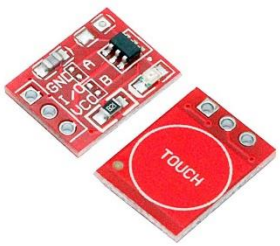

Рисунок 1.5 – *TTP*223

Система живлення є критично важливим компонентом, що забезпечує стабільну роботу всіх елементів пристрою. Особливу увагу приділено розрахунку споживання струму світлодіодною стрічкою, оскільки при максимальній яскравості всіх 29 світлодіодів загальний струм може досягати значних величин. Кожен світлодіод *WS*2812*B* при максимальній яскравості білого кольору споживає до 60 мА, що для 29 елементів становить до 1.74 А. З урахуванням споживання *ESP*8266 (близько 170 мА в активному режимі *Wi-Fi*) та *OLED*-дисплея (до 20 мА), загальне споживання може досягати 2А.

1.4.2 Технологічний стек для проксі-сервісу

Архітектура системи передбачає використання проміжного програмного компонента - проксі-сервісу, що виконує роль посередника між мікроконтролером *ESP*8266 та зовнішнім *API*, яке надає дані про повітряні тривоги. Необхідність створення такого проксі-сервісу зумовлена кількома технічними та функціональними вимогами, що не можуть бути ефективно вирішені безпосередньо на рівні мікроконтролера.

Першочерговою причиною є складність структури даних, що надаються оригінальним *API*. Зовнішні сервіси зазвичай повертають детальну інформацію у форматі *JSON*, що містить численні поля метаданих, вкладені об'єкти та масиви. Обробка таких структур на мікроконтролері з обмеженою оперативною пам'яттю та обчислювальними ресурсами є неефективною та може призводити до нестабільної роботи пристрою. Проксі-сервіс дозволяє виконати всю складну обробку на потужному сервері та передати на *ESP*8266 лише необхідні дані у максимально спрощеному форматі.

Другою важливою функцією проксі-сервісу є інкапсуляція логіки автентифікації та авторизації при роботі з зовнішніми *API*. Багато сучасних веб-сервісів використовують складні схеми автентифікації, такі як *OAuth* 2.0, *JWT* токени або *API* ключі з додатковими параметрами безпеки. Реалізація такої логіки на мікроконтролері значно ускладнила б прошивку та зменшила її надійність.

Проксі-сервіс бере на себе всю відповідальність за автентифікацію з зовнішніми сервісами, надаючи *ESP*8266 простий та захищений інтерфейс доступу.

Третьою ключовою перевагою є оптимізація мережевого трафіку. Оригінальні дані про повітряні тривоги можуть мати обсяг у кілька кілобайт, тоді як після обробки проксі-сервісом вони перетворюються у компактний 29-символьний рядок. Це радикально зменшує час завантаження даних на *ESP*8266, особливо важливо в умовах нестабільного інтернет-з'єднання, та знижує навантаження на бездротову мережу.

Для реалізації проксі-сервісу було обрано мову програмування *Python* версії 3.13. Цей вибір обґрунтовується кількома факторами, що роблять *Python* оптимальним для даного завдання. Насамперед, *Python* забезпечує виключну швидкість розробки завдяки простому та інтуїтивному синтаксису, що дозволяє швидко створювати та тестувати функціональність. Велика екосистема готових бібліотек для веб-розробки, роботи з *HTTP*-запитами, *JSON*-обробки та інших необхідних завдань мінімізує обсяг коду, що потребує написання з нуля.

Інтерпретована природа *Python* спрощує процес налагодження та модифікації сервісу без необхідності компіляції. Висока читабельність коду полегшує подальше супроводження та розширення функціональності. Крос-платформенність *Python* дозволяє розгортати сервіс на різних операційних системах без модифікацій.

В якості веб-фреймворку для створення *HTTP API* було обрано *FastAPI* - сучасний та високопродуктивний фреймворк для *Python*. *FastAPI* базується на стандартних підказках типів *Python* (*type hints*), що забезпечує автоматичну валідацію вхідних та вихідних даних за допомогою бібліотеки *Pydantic*. Це значно підвищує надійність коду та зменшує кількість потенційних помилок під час роботи з даними.

FastAPI забезпечує нативну підтримку асинхронного програмування за допомогою *async*/*await* конструкцій *Python*. Це критично важливо для веб-сервісу, що виконує мережеві запити до зовнішніх *API*, оскільки дозволяє обробляти множину одночасних запитів без блокування виконання. Асинхронна архітектура забезпечує високу продуктивність навіть при великому навантаженні.

Фреймворк також включає вбудовані засоби для автентифікації та авторизації, включаючи підтримку *JWT* токенів, *OAuth* та інших сучасних стандартів безпеки. Це дозволяє легко реалізувати захищений доступ до *API* сервісу.

Для виконання *HTTP*-запитів до зовнішніх *API* використовується бібліотека *requests* - де-факто стандарт для *HTTP*-клієнтів у *Python* екосистемі. *Requests* забезпечує простий та інтуїтивний інтерфейс для виконання *HTTP*-запитів різних типів, автоматичну обробку кодувань, підтримку *cookies*, сесій та автентифікації. Бібліотека відзначається високою надійністю та стабільністю, що критично важливо для сервісу, що працює в *production* середовищі.

Керування залежностями проєкту та віртуальними середовищами здійснюється за допомогою інструменту *uv* - сучасного та швидкого менеджера пакетів для *Python*. *UV* значно перевершує традиційні інструменти, такі як *pip* та *virtualenv*, за швидкістю встановлення пакетів та створення віртуальних середовищ. Інструмент забезпечує детерміністичне встановлення залежностей, що гарантує однакове середовище розробки та *production*.

Безпека доступу до *IoT*-ендпоінтів проксі-сервісу реалізована через систему *HTTP Bearer* токенів. Кожен *ESP*8266 пристрій повинен передавати валідний токен у заголовку *Authorization* для отримання доступу до даних. Токени можуть мати обмежений термін дії та різні рівні доступу, що дозволяє гнучко управляти безпекою системи.

1.4.3 Джерела даних та методи їх обробки

Основним джерелом актуальної інформації про стан повітряних тривог в Україні для розробленої системи служить офіційне *API* сервісу *api.alerts.in.ua*, що надає достовірні та оперативні дані про активні тривоги в усіх регіонах країни [1]. Вибір цього конкретного джерела обґрунтований його високою надійністю та повнотою інформації, що критично важливо для системи, призначеної для інформування громадян про загрози безпеці.

API характеризується стабільним часом відгуку та високою доступністю, що забезпечує безперебійну роботу системи навіть в умовах підвищеного навантаження. Сервіс підтримує сучасні стандарти веб-*API*, включаючи *RESTful* архітектуру, *JSON* формат даних та *HTTP Bearer* токени для автентифікації, що спрощує інтеграцію та забезпечує безпеку доступу.

Конфігурація доступу до *API* реалізована на двох рівнях архітектури системи. На рівні проксі-сервісу *URL endpoint* та токен автентифікації зберігаються як змінні середовища *ALERTS_API_URL* та *ALERTS_API_TOKEN* відповідно, що дозволяє легко змінювати конфігурацію без модифікації коду. На рівні мікроконтролера *ESP*8266 аналогічні параметри можуть бути збережені в *EEPROM* пам'яті пристрою, при цьому значення за замовчуванням визначені у конфігураційному файлі як *DEFAULT_ALERT_API_URL* та *DEFAULT_ALERT_API_TOKEN*. Така дворівнева конфігурація забезпечує гнучкість системи та можливість роботи як через проксі-сервіс, так і в режимі прямого доступу до *API*. Мапу тривог показано на рисунку 1.6.

Рисунок 1.6 – Мапа тривог

Структура даних, що повертаються первинним *API*, представляє собою *JSON*-об'єкт, що містить детальну інформацію про кожну активну тривогу. Кожен запис включає унікальний ідентифікатор локації (*location_uid*), що дозволяє однозначно ідентифікувати географічний регіон, тип локації (*location_type*), що

може вказувати на область, місто або спеціальну адміністративну одиницю, тип тривоги (*alert_type*), що визначає характер загрози, часові мітки початку та можливого завершення тривоги, а також додаткову метаінформацію про джерело та достовірність повідомлення.

Наступний етап обробки полягає у фільтрації та класифікації активних тривог. Алгоритм аналізує кожну тривогу з отриманого списку, виділяючи значення *location_uid* та *alert_type* для подальшої обробки. Для кожного з 29 цільових регіонів України, що включають 25 областей та 4 спеціальні адміністративні одиниці (місто Київ, місто Харків, місто Запоріжжя та Криворізький район), система визначає відповідний статус на основі активних тривог.

Важливим компонентом обробки є система мапінгу географічних ідентифікаторів на позиції у результуючому рядку. Для областей України використовується мапінг *OBLAST_UID_TO_POSITION*, що встановлює відповідність між унікальними ідентифікаторами областей у вихідних даних *API* та їхніми позиціями в 29-символьному рядку результату. Аналогічно, для спеціальних адміністративних одиниць застосовується мапінг *ZONE_UID_TO_POSITION*. Такий підхід забезпечує точне та незмінне відображення географічної структури України у вихідних даних системи.

Особливу увагу приділено обробці ситуацій множинних одночасних тривог в одному регіоні. Коли для певної території активні кілька різних типів загроз, система застосовує алгоритм пріоритизації, реалізований у методі *get_alert_priority*. Алгоритм враховує рівень небезпеки різних типів тривог та обирає найкритичніший для відображення. Наприклад, повітряна тривога може мати вищий пріоритет за артилерійський обстріл, що забезпечує адекватне інформування користувачів про найбільш актуальні загрози.

Результатом усього процесу обробки є генерація компактного 29-символьного рядка, де кожна позиція відповідає конкретному регіону України згідно з попередньо встановленим порядком, а символ у цій позиції кодує поточний статус тривоги в регіоні. Наприклад, рядок

"*PPPPPPAAPPPPPPPPPPPPPPPPPP*" може означати, що в більшості регіонів немає активних тривог (символ *P* - "*peace*"), але в деяких областях діє повітряна тривога (символ *A* - "*air raid*"). Такий формат забезпечує мінімальний обсяг переданих даних - лише 29 байт замість потенційних кілобайт оригінальної *JSON* структури [2].

На стороні мікроконтролера *ESP*8266 реалізований клієнтський компонент системи обробки даних, відповідальний за отримання обробленого рядка статусів від проксі-сервісу та його перетворення у візуальне представлення на фізичній карті. Модуль *alert_manager* керує процесом отримання даних, виконуючи періодичні *HTTP* запити до *IoT*-ендпоінту проксі-сервісу з інтервалом, визначеним константою *ALERT_UPDATE_INTERVAL_MS* у конфігураційному файлі (типово 30 секунд). Цей інтервал збалансований між необхідністю актуальності інформації та мінімізацією навантаження на мережу та сервер.

Ключовим етапом обробки на стороні *ESP*8266 є трансформація позицій символів у рядку статусів у фізичні індекси світлодіодів на *WS*2812*B* стрічці. Цей процес реалізується через систему мапінгу, що встановлюється під час ініціалізації пристрою функцією *initializeApiToLedMapping*(). Мапінг базується на співставленні трьох масивів конфігурації: *API_OBLAST_NAMES*, що визначає порядок регіонів у рядку статусів від проксі-сервісу, *PHYSICAL_LED_REGION_NAMES*, що містить назви регіонів у порядку їх фізичного розташування на карті, та *PHYSICAL_LED_STRIP_INDICES*, що встановлює відповідність між логічними позиціями регіонів та фізичними індексами світлодіодів на стрічці.

Фінальний етап обробки даних включає інтерпретацію символьних кодів статусів тривог у конкретні кольори для відображення на світлодіодній стрічці. Модуль *led_manager* відповідає за цю трансформацію, використовуючи попередньо визначені константи кольорів, такі як *COLOR_NO_ALERT_GREEN* для мирного стану, *COLOR_ALERT_FULL_RED* для повітряної тривоги, *COLOR_ARTILLERY_ORANGE* для артилерійського обстрілу тощо.

1.5 Проєктування архітектури системи

Архітектурне проєктування системи візуалізації повітряних тривог України представляє собою комплексний процес структурування та організації взаємодії компонентів, який визначає принципи функціонування та забезпечує ефективне виконання поставлених завдань. Розроблена архітектура базується на розподіленому підході, що дозволяє оптимально розподілити навантаження між різними рівнями системи та забезпечити високу гнучкість і масштабованість рішення.

Основу архітектурного рішення становить триярусна модель, що включає фізичний пристрій візуалізації, проміжний сервіс обробки даних та зовнішнє джерело інформації про тривоги. Така структура забезпечує чітке розмежування функціональних обов'язків між компонентами та створює надійну основу для подальшого розвитку системи. Кожен рівень архітектури має власну зону відповідальності, що спрощує розробку, тестування та підтримку системи в цілому.

Перший рівень архітектури представлений фізичним пристроєм, побудованим на основі мікроконтролера *ESP*8266. Цей компонент виконує функції безпосередньої взаємодії з користувачем та візуального представлення інформації про стан повітряних тривог. Архітектура програмного забезпечення пристрою побудована за модульним принципом, де кожен модуль інкапсулює специфічну функціональність. Модуль управління мережевими підключеннями забезпечує встановлення та підтримку з'єднання з локальною *Wi-Fi* мережею, автоматичне відновлення підключення у випадку збоїв та моніторинг якості сигналу. Модуль керування дисплеєм відповідає за виведення службової інформації на *OLED*-екран, включаючи поточний час, статус підключення, *IP*-адресу пристрою та додаткові діагностичні дані.

Система індикації реалізована через спеціалізований модуль управління світлодіодною стрічкою, який забезпечує точне відображення стану тривог для кожного регіону України. Цей модуль підтримує адресування окремих світлодіодів, що дозволяє створювати детальну карту стану безпеки по всій території країни. Алгоритми обробки включають функції плавного переходу між

кольорами, регулювання яскравості та реалізацію спеціальних режимів індикації для критичних ситуацій.

Модуль енергонезалежного зберігання налаштувань використовує вбудовану *EEPROM* пам'ять для збереження конфігураційних параметрів між сеансами роботи. Це включає параметри підключення до *Wi-Fi* мережі, налаштування взаємодії з проксі-сервісом, персональні налаштування користувача та калібрувальні дані для різних компонентів системи. Архітектура модуля передбачає версіонування структури даних, що забезпечує сумісність при оновленнях прошивки.

Веб-сервер, вбудований у прошивку *ESP*8266, реалізує інтерфейс для локального налаштування пристрою. Архітектура веб-сервера включає систему маршрутизації запитів, обробку статичних ресурсів, реалізацію *API* для зміни налаштувань та систему автентифікації для захисту від несанкціонованого доступу. Інтерфейс побудований з використанням адаптивного дизайну, що забезпечує коректне відображення на різних пристроях.

Другий рівень архітектури представлений проксі-сервісом, який виконує роль інтелектуального посередника між фізичними пристроями та зовнішніми джерелами даних. Архітектура сервісу побудована на основі мікросервісного підходу з чітким розділенням відповідальності між компонентами. Модуль маршрутизації запитів забезпечує обробку вхідних *HTTP*-запитів та їх направлення до відповідних обробників. Система підтримує версіонування *API*, що дозволяє поступово впроваджувати нові функції без порушення сумісності з існуючими пристроями.

Третій рівень архітектури представлений зовнішнім *API* постачальника даних про повітряні тривоги. Взаємодія з цим рівнем організована через спеціалізований модуль інтеграції, який забезпечує надійне отримання даних, обробку помилок та відновлення після збоїв. Така архітектурна модель забезпечує високу надійність системи за рахунок розподілення ризиків між компонентами та можливості незалежного масштабування кожного рівня.

РОЗДІЛ 2
РОЗРОБКА АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Розробка апаратної частини

2.1.1 Електрична схема

Електрична схема пристрою базується на мікроконтролерному модулі *ESP*8266, реалізованому на відлагоджувальних платах типу *NodeMCU*. Вибір цих плат обумовлений їхньою доступністю на ринку, наявністю інтегрованого *USB-UART* конвертера для спрощення процесу програмування та відлагодження, а також вбудованого стабілізатора напруги на 3.3В, що суттєво спрощує організацію живлення як самого мікроконтролера, так і підключених периферійних пристроїв.

Підключення адресної світлодіодної стрічки *WS*2812*B* розраховано на використання 29 світлодіодних елементів, що відповідають 25 обласним центрам України та 4 спеціальним зонам (м. Київ як окрема адміністративна одиниця, м. Харків, м. Запоріжжя, Криворізький район). Організація живлення стрічки потребує особливої уваги через значне споживання струму - кожен світлодіод може споживати до 60 мА при відображенні білого кольору на максимальній яскравості. Для 29 світлодіодів це може становити до 1.74А.

Критично важливим є забезпечення спільної землі між *ESP*8266 та джерелом живлення стрічки для уникнення проблем з передачею цифрових даних. При використанні довгих стрічок рекомендується підключати живлення з обох кінців або в декількох точках для забезпечення рівномірного розподілу напруги.

Інформаційний вхід *DIN* першого світлодіода стрічки підключається до одного з цифрових *GPIO* пінів *ESP*8266, визначеного як *LED_PIN* у конфігураційному файлі [3].

OLED-дисплей *SSD*1306 з роздільною здатністю 128*x*64 пікселі підключається через інтерфейс *I*2*C*. Лінія *SCL* дисплея з'єднується з піном *D*1 (*GPIO*5) *ESP*8266, а лінія даних *SDA* підключається до піна *D*2 (*GPIO*4). Живлення дисплея здійснюється від виходу 3.3В плати *ESP*8266, при цьому більшість модулів *OLED* вже містять необхідні підтягуючі резистори для ліній *I*2*C* на своїй платі.

Сенсорна кнопка *TTP*223 підключається таким чином: сигнальний вихід *I*/*O* кнопки з'єднується з цифровим піном *ESP*8266, визначеним як *TOUCH_BUTTON_PIN* у конфігурації. Живлення модуля кнопки здійснюється від 5В. Загальна організація живлення системи передбачає подачу основного живлення на вхід *Vin* плати *ESP*8266. *ESP*8266 через свій вбудований стабілізатор забезпечує живлення 3.3В для *OLED*-дисплея. Схему підключення компонентів показано на рисунку 2.1.

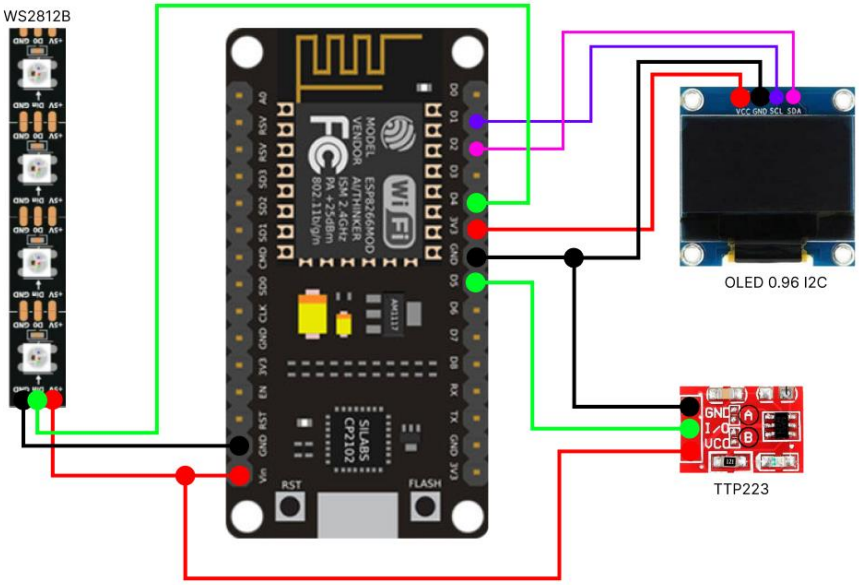

Рисунок 2.1 – Схема підключення компонентів

Компонувальне рішення передбачає інтеграцію всіх електронних компонентів у фізичну модель карти України. Світлодіоди монтуються у попередньо просвердлені отвори відповідного діаметра. Модуль *ESP*8266, *OLED*-дисплей та сенсорна кнопка розташовуються у легкодоступному для користувача місці. Всі з'єднувальні проводи акуратно прокладаються на зворотній стороні макета.

2.1.2 Збирання фізичної моделі

Процес збирання фізичної моделі вимагає ретельного планування та точного виконання кожного етапу. Спочатку на підготовленій основі карти проводиться

розмітка та свердління отворів для встановлення світлодіодів згідно з географічними координатами обласних центрів. *WS*2812*B* нарізається на окремі сегменти з урахуванням необхідної довжини проводів для кожного з'єднання. До кожного світлодіодного сегмента припаюються три основні проводи: *VCC* для живлення, *GND* для землі та *DIN* для входу даних, а також *DOUT* для виходу даних, якщо світлодіод не є останнім у ланцюгу. Використовуються якісний флюс та припій з відповідною температурою паяльника.

Після підготовки всіх світлодіодних елементів вони встановлюються на призначені місця на карті. З'єднання між світлодіодами виконуються згідно з визначеною послідовністю, при цьому вихід *DOUT* попереднього світлодіода з'єднується з входом *DIN* наступного, формуючи безперервний ланцюг передачі даних.

Виконання електричних з'єднань проводиться строго згідно з розробленою електричною схемою: лінія даних першого світлодіода підключається до *LED_PIN*, *OLED*-дисплей підключається до відповідних *I2C* пінів, сенсорна кнопка з'єднується з *TOUCH_BUTTON_PIN*. Окремо організовується живлення *ESP*8266 та світлодіодної стрічки з обов'язковим забезпеченням спільної землі між усіма компонентами системи. Підключення встановлених компонентів показано на рисунку 2.2.

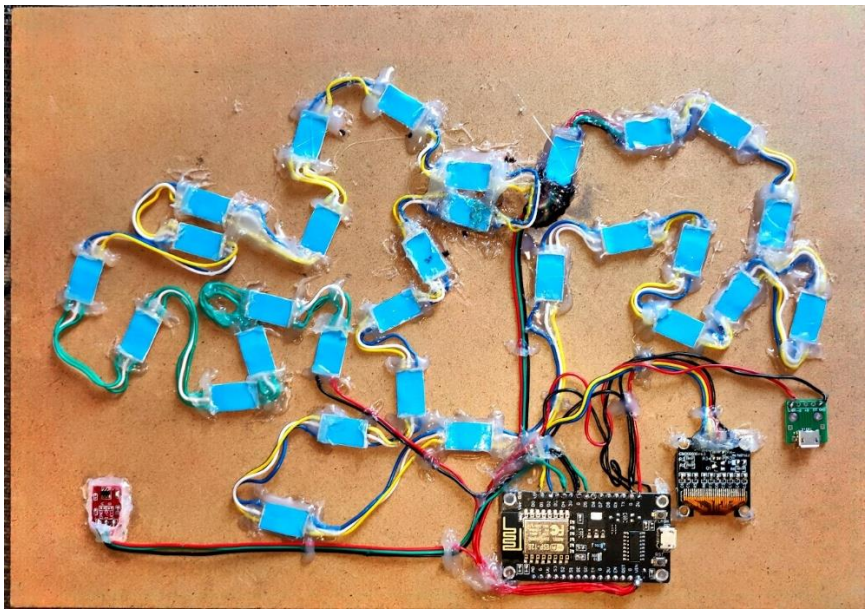

Рисунок 2.2 – Підключення встановлених компонентів

Перед першим увімкненням проводиться ретельна візуальна перевірка всіх паяних з'єднань на відсутність коротких замикань, обривів та якісних паяних контактів. За допомогою цифрового мультиметра перевіряються всі лінії живлення, вимірюється опір між силовими шинами та сигнальними лініями для виявлення можливих помилок монтажу. Макет карти тривог показано на рисунку 2.3.

Рисунок 2.3 – Макет карти тривог

Тестування апаратної частини починається з увімкнення живлення без завантаження основної прошивки.

2.2 Програмне забезпечення *ESP*8266

2.2.1 Структура прошивки та основні модулі

Прошивка для *ESP*8266 розроблена в середовищі *Arduino IDE* та має чітко структуровану архітектуру, що полегшує розуміння логіки роботи, внесення змін та розширення можливостей системи. Головний файл прошивки *esp-alertmap.ino* містить дві основні функції.

Функція *setup*() виконується однократно при старті мікроконтролера та відповідає за ініціалізацію всіх підсистем пристрою.

Ініціалізація *OLED*-дисплея супроводжується виведенням стартового екрана з назвою проєкту, що інформує користувача про початок роботи системи. Паралельно ініціалізується адресна світлодіодна стрічка з встановленням початкової яскравості згідно з збереженими налаштуваннями. Функцію *setup* показано на рисунку 2.4.

```
     void setup()
     {
       Serial.begin(115200);
       Serial.println(F("\nЗавантаження AlertMap..."));
       // Serial.printf_P(PSTR("Setup: Initial free heap: %u\n"),
ESP.getFreeHeap()); // Reduced logging

       pinMode(TOUCH_BUTTON_PIN, INPUT);
       DisplayManager::init();

       EEPROMManager::init();
       EEPROMManager::loadWiFiCredentials(savedSSID,
savedPassword);       // Load WiFi credentials
       EEPROMManager::loadApiConfig(savedAlertApiUrl,
savedAlertApiToken); // Load API config
       ledBrightness = EEPROMManager::loadLEDBrightness();
       noAlertDisplayMode =
EEPROMManager::loadNoAlertDisplayMode();

       LEDManager::init();
       initializeApiToLedMapping();
       AlertManager::init();

       if (savedSSID.length() > 0 &&
WiFiManager::connectToWiFi(savedSSID, savedPassword))
       {
         isAPMode = false;
         MDNSManager::init();
         timeClient.begin();
         Serial.println(F("Підключено до WiFi. NTP клієнт
запущено."));
       }
       else
       {
         WiFiManager::startAccessPoint();
       }

       WebServerManager::init();
       Serial.println(F("Веб-сервер запущено."));
     }
```

Рисунок 2.4 – Функція *setup*

Важливим етапом є виклик функції *initializeApiToLedMapping*(), яка створює таблицю відповідності між індексами регіонів у відповідях *API* та фізичними позиціями світлодіодів на стрічці. Функції *initializeApiToLedMapping* показано на рисунку 2.5.

```
     void initializeApiToLedMapping()
     {
       for (int i = 0; i < 27; ++i)
       {
         finalApiIndexToLedStripIndexMap[i] = -1;
       }
       for (int api_idx = 0; api_idx < 27; ++api_idx)
       {
         const char *api_oblast_name = API_OBLAST_NAMES[api_idx];
         bool mapped = false;
         for (int physical_map_idx = 0; physical_map_idx < 29;
++physical_map_idx)
         {
           if (strcmp(api_oblast_name,
PHYSICAL_LED_REGION_NAMES[physical_map_idx]) == 0)
           {
             finalApiIndexToLedStripIndexMap[api_idx] =
PHYSICAL_LED_STRIP_INDICES[physical_map_idx];
             mapped = true;
             break;
           }
         }
         if (!mapped && api_idx != 18)
         {
           Serial.printf_P(PSTR("Увага: API регіон '%s' (API індекс
%d) не вдалося зіставити з LED.\n"), api_oblast_name, api_idx);
         }
       }
     }
```

Рисунок 2.5 – Функції *initializeApiToLedMapping*

Після ініціалізації апаратних компонентів система намагається підключитися до *Wi-Fi* мережі, використовуючи збережені облікові дані.

Функція *loop*() виконується в нескінченному циклі після завершення *setup*() та містить основну логіку роботи пристрою. Тут обробляються *HTTP*-запити клієнтів до веб-сервера, оновлюється стан *mDNS*-сервісу, актуалізується інформація на *OLED*-дисплеї та обробляються натискання сенсорної кнопки користувачем. Функцію *loop* показано на рисунку 2.6.

```
    void loop()
    {
      unsigned long currentTime = millis();

      if (currentTime - lastHeapLogTime >= 60000)
      {
        Serial.printf_P(PSTR("Loop: Вільна пам'ять: %u. Час
роботи: %lu ms\n"), ESP.getFreeHeap(), currentTime);
        lastHeapLogTime = currentTime;
      }

      handleTouchButton();
      WebServerManager::handleClient();

      if (!isAPMode)
      {
        MDNSManager::update();
      }

      DisplayManager::update();

      if (WiFiManager::isConnected() && !isAPMode)
      {
        if (currentTime - lastAlertFetchTime >=
ALERT_UPDATE_INTERVAL_MS || lastAlertFetchTime == 0)
        {
          String newAlerts = AlertManager::fetchAlerts();
          if (newAlerts.length() == 27)
          {
            LEDManager::setAlertData(newAlerts);
          }
          else
          {
            LEDManager::setAlertData("");
          }
          lastAlertFetchTime = currentTime;
        }
      }

      LEDManager::update();
      WiFiManager::checkAPTimeout();

      if (!isAPMode && !WiFiManager::isConnected())
      {
        delay(3000);
        ESP.restart();
      }

      delay(50);
    }
```

Рисунок 2.6 – Функція *loop*

Структура програмного забезпечення включає декілька спеціалізованих модулів, кожен з яких інкапсулює певну функціональність. Модуль *config.h* визначає всі глобальні константи системи, включаючи номери пінів підключення компонентів, розміри буферів для обробки даних, параметри мережевого підключення за замовчуванням, *URL*-адреси та токени доступу до *API*, масиви імен регіонів для створення відповідності між *API* та фізичними світлодіодами. Тут же підключаються всі необхідні бібліотеки та оголошуються глобальні змінні для зберігання поточних налаштувань системи.

Модуль *eeprom_manager.h* реалізує всю логіку роботи з енергонезалежною пам'яттю *EEPROM* для зберігання користувацьких налаштувань. Він надає функції для збереження та завантаження облікових даних *Wi-Fi*, налаштувань режиму *API*, деталей підключення до офіційного та проксі *API*, налаштувань яскравості світлодіодів та режиму відображення стану без тривоги.

Менеджер *Wi-Fi* підключення *wifi_manager.h* відповідає за всі аспекти мережевого підключення, включаючи роботу в режимі станції для підключення до існуючої мережі, створення власної точки доступу для первісного налаштування та моніторинг стану з'єднання з автоматичним відновленням у разі втрати зв'язку.

Веб-сервер реалізований у модулі *web_server_manager.h* та забезпечує повнофункціональний інтерфейс для налаштування всіх параметрів системи. Він генерує динамічні *HTML*-сторінки з формами для введення налаштувань, обробляє запити на збереження конфігурації, сканування доступних *Wi-Fi* мереж, перезавантаження системи та скидання до заводських налаштувань.

Модуль *mdns_manager.h* налаштовує та підтримує роботу *multicast DNS* сервісу, що дозволяє користувачам звертатися до веб-інтерфейсу пристрою за зрозумілою назвою замість *IP*-адреси. Менеджер дисплея *display_manager.h* керує *OLED*-екраном, відображаючи поточний час, дату, *IP*-адресу, статус підключення та різноманітні інформаційні повідомлення для користувача [4].

Центральним компонентом візуалізації є модуль *led_manager.h*, який керує адресною світлодіодною стрічкою, інтерпретуючи отримані дані про стан тривог та встановлюючи відповідні кольори та яскравість для кожного регіону. Цей

модуль також реалізує різноманітні анімації для індикації стану завантаження, очікування даних та інших системних повідомлень.

Модуль *alert_manager.h* інкапсулює всю логіку отримання даних про тривоги з різних джерел. Він містить функціональність для формування *HTTP* та *HTTPS* запитів до офіційного *IoT API* або власного проксі-сервера, обробки отриманих відповідей та валідації даних перед передачею їх для візуалізації.

2.2.2 Алгоритми обробки тривог та керування індикацією

Процес починається з періодичного виклику функції *fetchAlerts*() менеджера тривог, яка спочатку аналізує збережений користувацький вибір джерела даних. Ця інформація зберігається в глобальній змінній *savedSelectedApiMode*, яка завантажується з *EEPROM* при старті системи або оновлюється після внесення змін через веб-інтерфейс.

Для офіційного *IoT API* система використовує збережені *URL*-адресу та токен доступу, формуючи *HTTPS GET*-запит до серверів держаних служб. Цей режим повертає текстовий рядок довжиною 27 символів, де кожен символ відповідає одній з областей України. Формат відповіді є спрощеним та містить лише базову інформацію про наявність або відсутність повітряної тривоги, використовуючи символи «*A*» для активної тривоги, «*P*» для часткової тривоги та «*N*» для відсутності тривоги.

Проксі *API* режим використовує власний сервер-посередник, що дозволяє отримувати розширену інформацію про стан безпеки. У цьому режимі система формує запит до користувацького проксі-сервера, використовуючи відповідні *URL* та токен доступу. Відповідь містить рядок довжиною 29 символів, що включає не лише 25 обласні центри, але й додаткові спеціальні зони: місто Київ як окрему адміністративну одиницю, міста Харків і Запоріжжя, Криворізький район.

Розширений формат проксі *API* підтримує різноманітні типи загроз, використовуючи символи «*P*» для мирного стану, «*A*» для повітряної тривоги, «*S*» для артилерійської загрози, «*F*» для активних бойових дій, «*C*» для хімічної загрози

та «*N*» для ядерної загрози. Така деталізація дозволяє користувачам отримувати більш повну картину безпекової ситуації в кожному регіоні.

Після отримання даних від будь-якого з *API* проводиться валідація формату відповіді, включаючи перевірку довжини рядка та наявності допустимих символів. У разі виявлення помилок мережі, невірного *HTTP*-коду відповіді або невалідного формату даних функція повертає порожній рядок або спеціальний маркер помилки для подальшої обробки системою індикації.

Валідні дані передаються до менеджера світлодіодів через функцію *setAlertData*(), яка зберігає отриманий рядок та встановлює прапорець активності даних. Оновлення візуальної індикації відбувається в функції *update*() менеджера світлодіодів, яка викликається в основному циклі програми. Функцію *setAlertData* показано на рисунку 2.7.

```
    void LEDManager::setAlertData(const String &alertStatusString)
    {
        if (alertStatusString.length() == 27)
        {
            s_currentAlertData = alertStatusString;
            s_alertsActive = true;
        }
        else
        {
            if (!alertStatusString.isEmpty())
            {
                Serial.print(F("LEDManager: Invalid alert data
received, length: "));
                Serial.println(alertStatusString.length());
            }
            s_currentAlertData = "";
            s_alertsActive = false;
        }
    }
```

Рисунок 2.7 – Функція *setAlertData*

При обробці 27-символьного рядка від офіційного *IoT API* система ітерує по кожному символу, використовуючи індекс для визначення відповідного фізичного світлодіода через таблицю відповідності *finalApiIndexToLedStripIndexMap*. Ця таблиця попередньо заповнюється при ініціалізації системи та забезпечує

правильну відповідність між порядком регіонів у *API* та їх географічним розташуванням на фізичній карті.

Для спеціальних зон, які мають власні світлодіоди на карті, але не представлені окремо в 27-символьному рядку, застосовується логіка успадкування стану від відповідних областей. Місто Київ успадковує статус від столичного регіону, Харків - від Харківської області, Запоріжжя - від Запорізької області, а Криворізький район - від Дніпропетровської області.

Обробка 29-символьного рядка від проксі *API* є більш прямолінійною, оскільки кожен символ безпосередньо відповідає фізичному світлодіоду на карті. Система послідовно обробляє кожен символ, встановлюючи відповідний колір згідно з типом загрози: зелений або вимкнений стан для мирного часу, червоний для повітряної тривоги, помаранчевий для артилерійської загрози, жовтий для активних бойових дій, фіолетовий для хімічної загрози та яскраво-червоний або малиновий для критичних станів.

Після встановлення кольорів для всіх світлодіодів викликається функція *show*() для фізичного оновлення стану стрічки. Загальна яскравість стрічки контролюється користувацьким налаштуванням *ledBrightness*.

2.2.3 Веб-інтерфейс для налаштування

Веб-інтерфейс є ключовим компонентом системи, що забезпечує зручний спосіб конфігурації пристрою через стандартний браузер. Реалізований у модулі *WebServerManager*, цей інтерфейс надає користувачеві повний контроль над усіма аспектами роботи системи моніторингу повітряних тривог.

Архітектурно веб-інтерфейс побудований як односторінковий додаток з динамічними елементами, що забезпечує інтуїтивну взаємодію з користувачем. Головна сторінка доступна за адресою кореневого шляху і містить всі необхідні секції для налаштування різних аспектів роботи пристрою.

Центральною особливістю інтерфейсу є секція налаштування джерела даних *API*, яка дозволяє користувачеві обирати між двома принципово різними режимами роботи системи. Перший режим передбачає використання офіційного *IoT API*, який

має обмежену функціональність через технічні обмеження мікроконтролера *ESP*8266. У цьому режимі система може отримувати лише базову інформацію про повітряні тривоги на рівні областей України, без деталізації за типами загроз та зональною системою.

Другий режим використовує спеціально розроблений проксі *API*, який дозволяє обійти технічні обмеження *ESP*8266 шляхом попередньої обробки повних даних на сервері. Проксі-сервіс отримує повну інформацію від офіційного *API*, обробляє її та надсилає мікроконтролеру у спрощеному форматі, що містить всі необхідні деталі про різні типи тривог та зональну систему попередження.

Секція відображення статусу надає користувачеві актуальну інформацію про поточний стан пристрою, включаючи параметри *Wi-Fi* підключення, *IP*-адресу, режим роботи (точка доступу або клієнт), а також *mDNS*-ім'я для зручного доступу в локальній мережі. Ця інформація автоматично оновлюється при завантаженні сторінки і дозволяє швидко діагностувати стан підключення. Веб-інтерфейс карти тривог показано на рисунку 2.8.

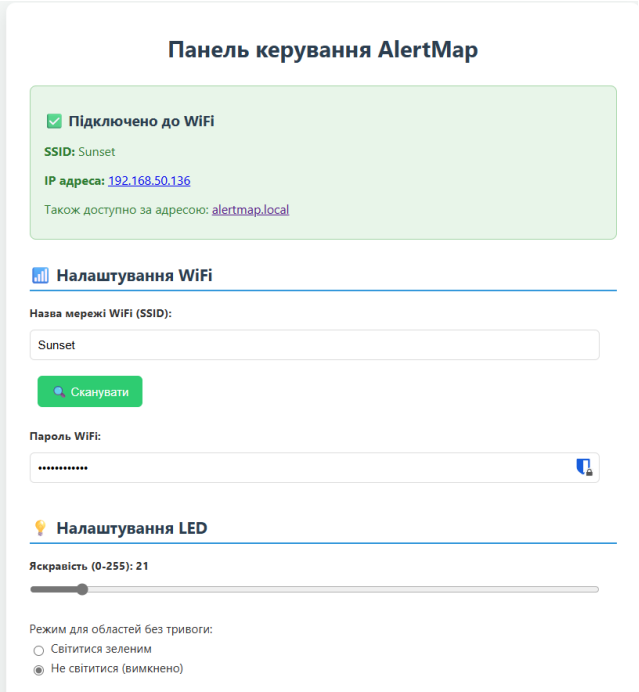

Рисунок 2.8 – Веб-інтерфейс карти тривог

Налаштування *Wi-Fi* реалізовані через інтерактивну секцію, що включає поле для введення *SSID* мережі та механізм автоматичного сканування доступних точок

доступу. Кнопка «Сканувати мережі» ініціює асинхронний *JavaScript*-запит на ендпоінт /*scan*, який повертає *JSON*-список доступних *Wi-Fi* мереж. Результати сканування відображаються у вигляді інтерактивного списку, де клік по назві мережі автоматично заповнює відповідне поле вводу. Це значно спрощує процес підключення до нової мережі, особливо в середовищах з великою кількістю доступних точок доступу.

Секція налаштування *LED* індикації містить елементи управління візуальним відображенням стану системи. Повзунок для регулювання яскравості реалізований через *HTML5* елемент *input* типу *range* з діапазоном значень від 0 до 255, що відповідає повному діапазону ШІМ-модуляції *ESP*8266. *JavaScript* забезпечує динамічне відображення поточного значення яскравості поруч з повзунком, надаючи користувачеві миттєвий зворотний зв'язок при налаштуванні.

Блок налаштувань офіційного *IoT API* містить поля для введення *URL* та токена доступу. Ці поля попередньо заповнюються збереженими значеннями або значеннями за замовчуванням з файлу конфігурації. Користувач може змінити ці параметри для використання альтернативних ендпоінтів або оновлення токенів доступу.

Кнопка «Зберегти всі налаштування» ініціює *POST*-запит на ендпоінт /*save*, який обробляє всі отримані дані форми. Обробник проводить базову валідацію введених даних, включаючи перевірку на порожні *URL*-адреси для обраного режиму *API* та коректність формату введених параметрів. Налаштування показано на рисунку 2.9.

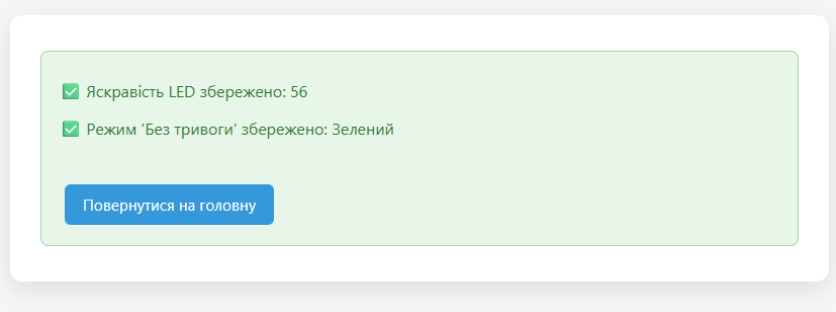

Рисунок 2.9 – Налаштування

Після успішної валідації система оновлює глобальні змінні в оперативній пам'яті *ESP*8266 та викликає відповідні функції *EEPROMManager* для збереження кожної групи налаштувань у енергонезалежну пам'ять. Процес збереження організований по групах: налаштування *Wi-Fi*, режим *API*, параметри офіційного *API*, параметри проксі *API* та налаштування *LED* індикації.

Система автоматично визначає чи потребують внесені зміни перезавантаження пристрою. Критичні зміни, такі як параметри *Wi-Fi* або *URL API*, встановлюють відповідний прапорець, що ініціює процедуру перезавантаження після збереження налаштувань.

Секція обслуговування пристрою надає доступ до сервісних функцій системи. Посилання на оновлення прошивки веде на спеціалізований ендпоінт /*update*, що реалізує функціональність *Over-The-Air* (*OTA*) оновлень. Це дозволяє оновлювати прошивку пристрою без фізичного доступу до нього, що особливо важливо для пристроїв, встановлених у важкодоступних місцях.

Кнопка перезавантаження надсилає запит на ендпоінт /*restart*, який ініціює програмне перезавантаження *ESP*8266. Це корисно для застосування змін, що вимагають повної ініціалізації системи або для відновлення роботи у випадку нестабільної поведінки.

Функція скидання до заводських налаштувань реалізована через ендпоінт /*reset* з додатковим підтвердженням через *JavaScript* функцію *confirm*().

Весь інтерфейс розроблений з урахуванням принципів адаптивного дизайну, що забезпечує коректну роботу на різних пристроях, від настільних комп'ютерів до мобільних телефонів. *CSS*-стилі та *JavaScript*-код оптимізовані для мінімального споживання ресурсів *ESP*8266, при цьому зберігаючи функціональність та зручність використання.

2.3 Проксі-сервіс

2.3.1 *API* для взаємодії з пристроєм

Фізична модель карти може працювати у двох режимах: з використанням офіційного *IoT API* та через проксі-сервіс. Перший режим передбачає пряме

звернення *ESP*8266 до спеціального ендпоінта для *IoT*-пристроїв, що надається ресурсом *alerts.in.ua*. Однак цей режим має суттєві обмеження - він підтримує лише повітряні тривоги та надає інформацію виключно на рівні областей. Другий режим роботи через проксі-сервіс дозволяє використовувати повний функціонал офіційного *API*, забезпечуючи підтримку різних типів тривог та зональної системи.

Розробка спеціалізованого *API* для взаємодії з мікроконтролером *ESP*8266 є центральним елементом архітектури проксі-сервісу. Необхідність створення такого інтерфейсу зумовлена принциповими обмеженнями *ESP*8266 у обробці великих обсягів *JSON*-даних, що повертаються повним *API alerts.in.ua*. Офіційний *IoT*-ендпоінт, незважаючи на свою оптимізацію для пристроїв з обмеженими ресурсами, не може забезпечити необхідний рівень деталізації інформації про тривоги.

Проксі-сервіс виступає інтелектуальним посередником між офіційним *API* та мікроконтролером, беручи на себе складні операції отримання, обробки та агрегації даних про тривоги. Він перетворює детальну інформацію у компактний, легко засвоюваний формат, оптимізований специфічно для можливостей *ESP*8266. Такий підхід дозволяє значно розширити функціональні можливості візуалізації без перевантаження мікроконтролера обчислювальними завданнями.

Основним ендпоінтом для взаємодії з *ESP*8266 є *GET* /*api*/*v*1/*alerts*/*iot*. При зверненні до цього ендпоінта мікроконтролер отримує відповідь у вигляді простого текстового рядка довжиною 29 символів. Кожен символ у цьому рядку відповідає поточному статусу тривоги для конкретного регіону або зонального об'єкта України. Такий формат мінімізує обсяг переданих даних та максимально спрощує їх парсинг на стороні мікроконтролера.

Система кодування статусів тривог використовує наступні символи: *P* для позначення відсутності тривоги (*Peace*), *A* для повітряної тривоги (*Air Raid*), *S* для загрози артилерійського обстрілу (*Artillery Shelling*), *F* для загрози вуличних боїв (*Urban Fights*), *C* для хімічної загрози (*Chemical*), та *N* для ядерної загрози (*Nuclear*). Кожен символ однозначно ідентифікує найвищий пріоритет тривоги для відповідного регіону.

Структура 29-символьного рядка чітко визначена та оптимізована під фізичну конфігурацію карти. Перші 25 позицій відповідають усім областям України, включаючи Автономну Республіку Крим. Наступні 4 позиції представляють спеціальні зональні об'єкти, які можуть мати власний статус тривоги незалежно від області, до якої вони адміністративно належать. До цих об'єктів належать місто Київ, місто Харків, місто Запоріжжя та Криворізький район. Така структура забезпечує точну відповідність між отриманими даними та 29 світлодіодами на фізичній моделі карти.

2.3.2 Інтеграція з джерелами даних про тривоги

Процес інтеграції починається з автентифікації проксі-сервісу в системі *alerts.in.ua* за допомогою унікального *Bearer* токену. Цей токен отримується окремо для кожного проксі-сервісу та забезпечує авторизований доступ до повного *API*. Токен зберігається як змінна середовища *ALERTS_API_TOKEN* та використовується у всіх запитах до офіційного *API*. Документацію проксі-сервера показано на рисунку 2.10.

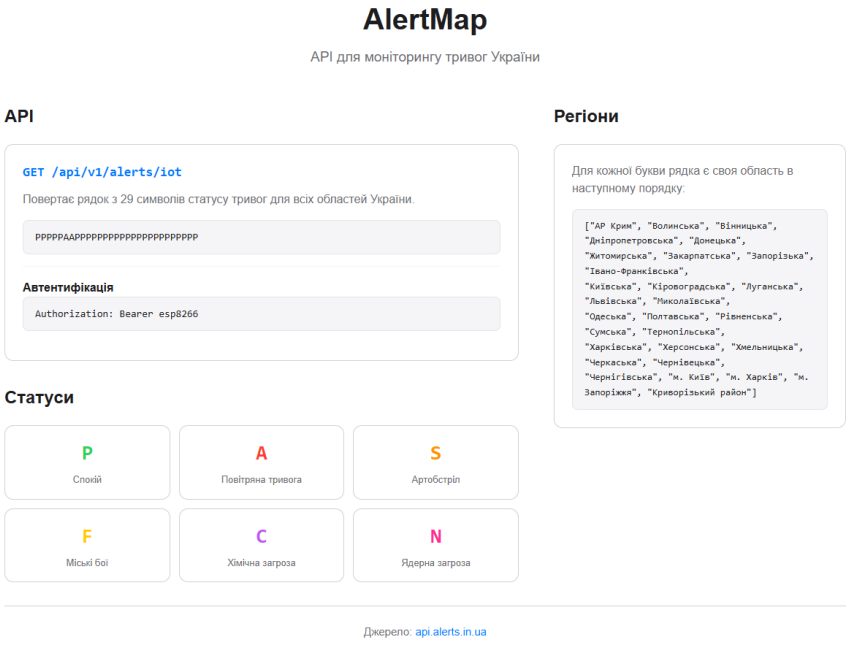

Рисунок 2.10 – Документація проксі-сервера

При отриманні запиту від *ESP*8266, проксі-сервіс ініціює *HTTP GET*-запит до повного ендпоінта *alerts.in.ua*. Відповідь надходить у форматі *JSON* та містить масив об'єктів, кожен з яких описує окрему активну тривогу. Кожен об'єкт тривоги включає детальну інформацію: тип тривоги (*alert_type*), унікальний ідентифікатор локації (*location_uid*), тип локації (*location_type*), назву локації, час початку тривоги та інші метадані.

Обробка отриманих даних починається з ініціалізації внутрішньої структури, що представляє статуси для всіх 29 контрольованих позицій. Початкове значення для кожної позиції встановлюється як "*P*", що означає відсутність тривоги. Далі система послідовно аналізує кожну тривогу з отриманого списку, визначаючи її тип та локацію.

Оскільки для одного регіону може бути активно декілька тривог різних типів, проксі-сервіс застосовує ієрархію пріоритетів для визначення найкритичнішого статусу. Пріоритети розташовані у наступному порядку (від найвищого до найнижчого): ядерна загроза, хімічна загроза, загроза вуличних боїв, загроза артилерійського обстрілу, повітряна тривога. Якщо для регіону вже встановлено тривогу певного рівня і надходить інформація про тривогу вищого пріоритету, статус оновлюється відповідним чином.

Визначення відповідності між локаціями з *API* та позиціями у 29-символьному рядку здійснюється через внутрішні мапінги. Система використовує два основні словники: *OBLAST_UID_TO_POSITION* для областей та *ZONE_UID_TO_POSITION* для спеціальних зональних об'єктів. Ці мапінги забезпечують точну відповідність між унікальними ідентифікаторами локацій з *API alerts.in.ua* та позиціями у результуючому рядку.

Після обробки всіх активних тривог система генерує фінальний 29-символьний рядок, використовуючи словник *ALERT_TYPE_TO_SYMBOL* для перетворення типів тривог у відповідні символи. Цей рядок і повертається мікроконтролеру *ESP*8266 як відповідь на його запит.

2.3.3 Розгортання сервісу

Розгортання проксі-сервісу має кілька варіантів реалізації. Найпрофесійнішим підходом є використання *Docker* з готовими *Dockerfile* та *docker-compose.yml* файлами. *Docker* забезпечує ізольоване середовище, що виключає конфлікти з іншими додатками, гарантує портативність між різними операційними системами та спрощує масштабування при зростанні навантаження. Конфігурація здійснюється через змінні середовища: *ALERTS_API_TOKEN* для авторизації в *API alerts.in.ua* та *ALERTS_API_URL* для базової адреси офіційного *API*.

Для розробки та тестування доступний локальний запуск безпосередньо в *Python*-середовищі. Процес включає клонування репозиторію, встановлення залежностей з файлу *pyproject.toml* та запуск через *Uvicorn* командою *uvicorn main:app --host* 0.0.0.0 *--port* 8000. Обов'язковим є наявність файлу *tokens.json* з авторизованими *Bearer* токенами для доступу до проксі-сервісу.

Проксі-сервіс можна успішно розгорнути на хмарних платформах: *Heroku* для швидкого розгортання, *AWS* з *Elastic Beanstalk* або *ECS*/*EKS*, *Google Cloud* з *Cloud Run* або *GKE*, *Microsoft Azure* з *App Service* або *AKS*. Вибір платформи залежить від вимог до продуктивності, бюджетних обмежень та потреб у масштабуванні.

Критично важливо забезпечити стабільний мережевий доступ для *ESP*8266 через *HTTP*/*HTTPS* протокол. При використанні хмарних платформ рекомендується налаштування *HTTPS* з валідними *SSL*-сертифікатами для безпеки. Необхідно передбачити моніторинг працездатності, логування помилок та автоматичний перезапуск у разі збоїв для забезпечення безперебійної роботи системи.

РОЗДІЛ 3
ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ

3.1 Методика тестування системи

Методика тестування системи фізичної моделі карти повітряних тривог України на базі мікроконтролера *ESP*8266 спрямована на всебічну перевірку працездатності, надійності та відповідності розробленого комплексу поставленим функціональним вимогам. Тестування проводилося на кількох рівнях, охоплюючи як окремі компоненти, так і систему в цілому.

Основні етапи тестування включали модульне тестування, інтеграційне тестування та системне тестування. Модульне тестування передбачало перевірку кожного апаратного та програмного компонента системи окремо для виявлення локальних дефектів. На цьому етапі тестувалася працездатність мікроконтролера *ESP*8266, адресної світлодіодної стрічки *WS*2812*B* з перевіркою кожного світлодіода та коректності відображення кольорів, *OLED*-дисплея *SSD*1306 з ініціалізацією та виведенням тексту і графіки, сенсорної кнопки *TTP*223 на реєстрацію натискань, модуля *Wi-Fi* у різних режимах роботи та *EEPROM* на запис і читання даних.

Тестове середовище включало спеціально підготовлену інфраструктуру з проксі-сервісом, розгорнутим локально. Для симуляції різних сценаріїв тривог використовувався розроблений емулятор *alertmap-proxy-emulator*, який дозволяв генерувати контрольовані відповіді *API*. Також активно використовувалися реальні дані з офіційного *API alerts.in.ua* для валідації. Моніторинг роботи *ESP*8266 здійснювався через *Serial Monitor* в *Arduino IDE*, а робота проксі-сервісу контролювалася через лог-файли та інструменти розробника веб-браузера.

Наприклад, для візуалізації тривоги критерієм було коректне засвічування відповідного світлодіода потрібним кольором протягом визначеного часу після отримання сигналу. Загальним критерієм успішності стало повне виконання всіх функціональних вимог, стабільна робота системи без збоїв протягом тривалого часу та коректна реакція на всі передбачені сценарії взаємодії.

3.2 Функціональне тестування компонентів

Тестування апаратної частини *ESP*8266 проводилося систематично для кожного модуля. Адресна світлодіодна стрічка *WS*2812*B* перевірялася шляхом послідовного ввімкнення та вимкнення кожного з 29 світлодіодів, а також встановлення основних кольорів включаючи червоний, зелений, синій, білий, жовтий, помаранчевий, фіолетовий та блакитний для перевірки коректності передачі кольору. Особлива увага приділялася тестуванню програмного керування яскравістю світлодіодів. *OLED*-дисплей *SSD*1306 проходив перевірку коректності ініціалізації, виведення текстової інформації різними шрифтами та розмірами, відображення спеціальних символів та функції очищення екрану. Відображення інформації на дисплеї показано на рисунку 3.1.

Рисунок 3.1 – Відображення інформації на дисплеї

Сенсорна кнопка *TTP*223 тестувалася на чітке спрацювання при натисканні, відсутність помилкових спрацювань та коректну передачу сигналу мікроконтролеру. Вбудований модуль *Wi-Fi* мікроконтролера *ESP*8266 проходив комплексне тестування здатності сканувати доступні *Wi-Fi* мережі, успішно підключатися до мереж з різними типами шифрування включаючи *WPA* та *WPA*2-*PSK*, а також до відкритих мереж. Перевірялася стабільність з'єднання та робота в режимі точки доступу з можливістю підключення клієнтських пристроїв.

EEPROM пам'ять тестувалася на коректність запису та зчитування усіх конфігураційних даних включаючи *SSID*, пароль, *URL API*, токен, налаштування яскравості, режим відображення областей без тривоги та режим роботи *API*. Критично важливою була перевірка збереження даних після повного вимкнення живлення пристрою та логіки магічного байта для визначення наявності збережених налаштувань.

Модуль отримання даних про тривоги тестувався на здатність формувати коректні *HTTP* та *HTTPS* запити до *API* як до офіційного *IoT*-ендпоінту, так і до проксі-сервісу. Перевірялася обробка успішних відповідей та відповідей з помилками включаючи 401 *Unauthorized*, 404 *Not Found*, 503 *Service Unavailable*, а також парсинг отриманого рядка статусу тривог довжиною 27 символів для прямого режиму та 29 символів для проксі-режиму.

Модуль веб-сервера проходив тестування доступності всіх сторінок веб-інтерфейсу та коректності відображення поточних налаштувань у полях форм. Перевірялася функція сканування та відображення списку доступних *Wi-Fi* мереж. Детально тестувалася логіка збереження всіх налаштувань включаючи параметри *Wi-Fi*, *URL* та токен *API* окремо для прямого та проксі режимів, яскравість *LED*, режим без тривоги та вибір режиму роботи *API*. Перевірялися функції перезавантаження пристрою та повного скидання налаштувань.

Проксі-сервіс *alertmap-proxy*, розроблений на *FastAPI*, також проходив функціональне тестування. Модуль автентифікації перевірявся на коректну валідацію *Bearer* токенів у заголовку *Authorization* з тестуванням сценаріїв з валідним токеном, невалідним токеном та відсутнім токеном. Обробка маршрутів тестувалася на правильну роботу основного ендпоінту /*api*/*v*1/*alerts*/*iot* з перевіркою повернення відповіді у форматі *PlainTextResponse* та коректної обробки помилок при взаємодії з сервісним шаром. Запити на проксі-сервер показано на рисунку 3.2.

```
INFO:app.auth:Успішна авторизація з токеном: 8bee6416f7...
INFO:app.services:Отримано 11 записів про тривоги
INFO:app.services:Знайдено 3 обласних тривог
INFO:app.services:Знайдено 1 зональних тривог
INFO:app.services:Результуючий рядок (29 символів): APPPPPPPPPPPAPPPPPPPAPPPPPPPAPP
INFO:     172.18.0.1:56538 - "GET /api/v1/alerts/iot HTTP/1.1" 200 OK
INFO:app.auth:Успішна авторизація з токеном: 8bee6416f7...
INFO:app.services:Отримано 11 записів про тривоги
INFO:app.services:Знайдено 3 обласних тривог
INFO:app.services:Знайдено 1 зональних тривог
INFO:app.services:Результуючий рядок (29 символів): APPPPPPPPPPPAPPPPPPPAPPPPPPPAPP
INFO:     172.18.0.1:56538 - "GET /api/v1/alerts/iot HTTP/1.1" 200 OK
```

Рисунок 3.2 – Запити на проксі-сервер

Конфігураційний модуль тестувався на коректне завантаження змінних середовища та файлу *tokens.json* для автентифікації запитів до самого проксі-сервісу. Емулятор проксі-сервісу *alertmap-proxy-emulator* тестувався на здатність генерувати різноманітні сценарії тривог включаючи різні типи та локації, а також на коректне завершення симульованих тривог з відповідністю формату *API*-відповіді очікуваному від реального *API alerts.in.ua*.

Успішне проходження всіх функціональних тестів окремих компонентів дозволило перейти до етапу інтеграційного тестування з більшою впевненістю у надійності системи та готовністю до перевірки взаємодії між компонентами.

3.3 Інтеграційне тестування та результати

Перший сценарій передбачав роботу *ESP*8266 в прямому режимі з офіційним *IoT API alerts.in.ua*. *ESP*8266 було налаштовано через веб-інтерфейс на прямий режим з *URL API https*://*api.alerts.in.ua*/*v*1/*iot*/*active_air_raid_alerts_by_oblast.json* та відповідним токеном доступу. Після підключення до *Wi-Fi* пристрій успішно надсилав запити до вказаного ендпоінту та отримував 27-символьний рядок з інформацією виключно про повітряні тривоги та їх відсутність на рівні областей. Світлодіоди на карті коректно відображали ці статуси з червоним кольором для активної повітряної тривоги та зеленим або вимкненим станом для спокійних регіонів залежно від налаштувань.

Зональні об'єкти включаючи місто Київ, Харків, Запоріжжя та Криворізький район успадковували статус відповідної області оскільки *IoT API* не надає для них окремої інформації. Оновлення статусів на карті відбувалося синхронно зі змінами даних в офіційному *API* з урахуванням встановленого інтервалу запитів. Результати цього тесту підтвердили, що пристрій може функціонувати автономно використовуючи обмежений, але офіційний канал даних, що є важливим для користувачів які не бажають розгортати власний проксі-сервіс. Обмеженням виявилася відсутність інформації про інші типи тривог та деталізації по зонах.

Другий сценарій передбачав роботу *ESP*8266 в режимі проксі з використанням *alertmap-proxy-emulator*. *ESP*8266 було переведено в режим проксі

через веб-інтерфейс з вказівкою *URL* локально розгорнутого проксі-сервісу, а сам проксі-сервіс був налаштований на отримання даних від емулятора. Емулятор дозволяв гнучко симулювати різноманітні ситуації включаючи виникнення повітряної тривоги в одній або декількох областях, активацію різних типів тривог включаючи артилерійський обстріл, міські бої, хімічну загрозу та ядерну загрозу як для областей так і для зональних об'єктів, одночасну тривогу в області та її адміністративному центрі, а також повну відсутність тривог.

*ESP*8266 успішно надсилав запити до проксі-сервісу, який у свою чергу отримував дані від емулятора, обробляв їх застосовуючи логіку пріоритетів при генерації кількох типів тривог для одного регіону та повертав *ESP*8266 коректний 29-символьний рядок. Світлодіоди на фізичній карті точно відображали симульовану ситуацію використовуючи відповідні кольори для кожного типу тривоги та для кожного з 29 регіонів включаючи 25 областей та 4 зональні об'єкти. Цей тест продемонстрував повну функціональність системи при роботі з розширеним набором даних що надається проксі-сервісом включаючи підтримку всіх типів тривог та зональної системи.

Третій сценарій був аналогічний попередньому але проксі-сервіс було переналаштовано на роботу з реальним повним *API alerts.in.ua* використовуючи відповідний токен доступу. Тестування проводилося шляхом спостереження за реальною ситуацією з тривогами в Україні. Фізична карта на *ESP*8266 коректно відображала актуальну інформацію що надходила від проксі-сервісу включаючи різні типи тривог та статуси для зональних об'єктів якщо такі були активні. Система демонструвала стабільну роботу протягом декількох годин безперервного моніторингу. Затримка між фактичним оголошенням тривоги та її відображенням на карті складалася з часу оновлення даних в *API alerts.in.ua*, часу обробки запиту проксі-сервісом та інтервалу запитів самого *ESP*8266 за замовчуванням 30 секунд.

Четвертий сценарій передбачав тестування функціональності веб-інтерфейсу *ESP*8266. Було перевірено всі функції конфігурування пристрою. Зміна налаштувань *Wi-Fi* включаючи *SSID* та пароль призводила до успішного перепідключення до нової мережі після збереження та перезавантаження. Зміна

URL API та токенів для обох режимів роботи прямого та проксі, а також перемикання між цими режимами коректно зберігалися в *EEPROM* і пристрій починав використовувати нові параметри для запитів що підтверджувалося аналізом мережевого трафіку та поведінкою *LED*-індикації.

Регулювання яскравості світлодіодів та вибір режиму без тривоги миттєво застосовувалися на карті. Функція *OTA*-оновлення прошивки була успішно протестована з завантаженням *bin* файлу нової версії прошивки через веб-інтерфейс, процес оновлення пройшов без помилок і пристрій перезавантажився з оновленою прошивкою зберігши попередні налаштування. Функція скидання налаштувань коректно повертала всі параметри до значень за замовчуванням і пристрій переходив у режим точки доступу.

Результати інтеграційного тестування показали що всі компоненти системи успішно взаємодіють між собою. Система коректно функціонує в обох передбачених режимах як при прямому підключенні до обмеженого *IoT API alerts.in.ua* так і при роботі через проксі-сервіс що дозволяє використовувати повний набір даних про типи тривог та зональну систему. Веб-інтерфейс надав зручний спосіб конфігурування всіх аспектів роботи пристрою включаючи вибір джерела даних. Функція *OTA*-оновлення значно спрощує процес оновлення програмного забезпечення пристрою.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно розроблено апаратно-програмний комплекс для візуалізації актуального стану повітряних тривог на фізичній карті України, що повністю відповідає поставленій меті та вирішує всі визначені завдання.

Проведений детальний аналіз предметної області інформування про повітряні тривоги дозволив виявити переваги та недоліки існуючих цифрових та традиційних засобів оповіщення.

Сформульовані функціональні вимоги до системи охопили всі ключові аспекти роботи: точне відображення стану тривог, надійне отримання даних з *API*, їх ефективну обробку, зручне користувацьке конфігурування через веб-інтерфейс, чітку індикацію стану пристрою та високу надійність роботи в цілому.

Вибір мікроконтролера *ESP*8266 як основи апаратної частини виявився оптимальним рішенням завдяки його інтегрованому *Wi-Fi* модулю, достатній продуктивності та доступності. Адресна світлодіодна стрічка *WS*2812*B* забезпечила яскраву та енергоефективну візуалізацію, *OLED*-дисплей *SSD*1306 надав можливість виведення детальної інформації, а сенсорна кнопка *TTP*223 створила зручний інтерфейс взаємодії.

Проміжний проксі-сервіс на *Python* успішно виконує функції посередника між повним *API alerts.in.ua* та *ESP*8266, ефективно обробляючи складні дані про різні типи тривог (повітряна, артобстріл, вуличні бої, хімічна, ядерна).

Розроблений апаратно-програмний комплекс представляє собою завершене та функціональне рішення, що забезпечує наочну та оперативну візуалізацію карти повітряних тривог України. Система характеризується високою гнучкістю у налаштуванні, підтримує оновлення програмного забезпечення та може слугувати надійним додатковим джерелом інформації про безпекову ситуацію для громадян, доповнюючи існуючі засоби оповіщення та підвищуючи загальний рівень інформованості населення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Devs Alerts in UA* – платформа сповіщень. [Електронний ресурс] – Режим доступу: *https*://*devs*.*alerts*.*in*.*ua*/ (дата звернення: 13.05.2025)

2. *ESP*8266*EX* – технічна документація. [Електронний ресурс] – Режим доступу: *https*://*www*.*espressif*.*com*/*sites*/*default*/*files*/*documentation*/0*a*-*esp*8266*ex_datasheet_en*.*pdf* (дата звернення: 27.05.2025)

3. *Adafruit NeoPixel* – посібник користувача. [Електронний ресурс] – Режим доступу: *https*://*learn*.*adafruit*.*com*/*adafruit*-*neopixel*-*uberguide* (дата звернення: 04.05.2025)

4. *Adafruit OLED* – бібліотека та приклади для *Arduino*. [Електронний ресурс] – Режим доступу: *https*://*learn*.*adafruit*.*com*/*monochrome*-*oled*-*breakouts*/*arduino*-*library*-*and*-*examples* (дата звернення: 18.05.2025)

5. *TTP*223 – технічна документація сенсора. [Електронний ресурс] – Режим доступу: *https*://*www*.*alldatasheet*.*com*/*datasheet*-*pdf*/*pdf*/533995/*TONTEK*/*TTP*223.*html* (дата звернення: 10.05.2025)

6. *Arduino* – довідник функцій. [Електронний ресурс] – Режим доступу: *https*://*www*.*arduino*.*cc*/*reference*/*en*/ (дата звернення: 25.05.2025)

7. *ESP*8266 *Arduino* – *GitHub* репозиторій. [Електронний ресурс] – Режим доступу: *https*://*github*.*com*/*esp*8266/*Arduino* (дата звернення: 02.05.2025)

8. *ArduinoJson* – офіційна документація. [Електронний ресурс] – Режим доступу: *https*://*arduinojson*.*org*/ (дата звернення: 20.05.2025)

9. *FastAPI* – офіційна документація. [Електронний ресурс] – Режим доступу: *https*://*fastapi*.*tiangolo*.*com*/ (дата звернення: 29.05.2025)

10. *Python* – офіційна документація. [Електронний ресурс] – Режим доступу: *https*://*docs*.*python*.*org*/ (дата звернення: 06.05.2025)

11. *Requests* – документація бібліотеки. [Електронний ресурс] – Режим доступу: *https*://*requests*.*readthedocs*.*io*/*en*/*latest*/ (дата звернення: 22.05.2025)

12. *Uvicorn* – офіційна документація. [Електронний ресурс] – Режим доступу: *https*://*www*.*uvicorn*.*org*/ (дата звернення: 01.05.2025)

13. *Docker* – офіційна документація. [Електронний ресурс] – Режим доступу: *https*://*docs*.*docker*.*com*/ (дата звернення: 15.05.2025)

14. *MDN Web Docs – HTML* документація. [Електронний ресурс] – Режим доступу: *https*://*developer*.*mozilla*.*org*/*en-US*/*docs*/*Web*/*HTML* (дата звернення: 12.05.2025)

15. *MDN Web Docs – CSS* документація. [Електронний ресурс] – Режим доступу: *https*://*developer*.*mozilla*.*org*/*en-US*/*docs*/*Web*/*CSS* (дата звернення: 08.05.2025)

16. *MDN Web Docs* – посібник з *JavaScript*. [Електронний ресурс] – Режим доступу: *https*://*developer*.*mozilla*.*org*/*en-US*/*docs*/*Web*/*JavaScript*/*Guide* (дата звернення: 17.05.2025)

17. *NTPClient* – бібліотека *Arduino*. [Електронний ресурс] – Режим доступу: *https*://*www*.*arduino*.*cc*/*reference*/*en*/*libraries*/*ntpclient* (дата звернення: 11.05.2025)

ДОДАТОК А

Головний файл робочої програми *esp-alertmap.ino*

```
#include "config.h"
#include "eeprom_manager.h"
#include "wifi_manager.h"
#include "web_server_manager.h"
#include "mdns_manager.h"
#include "display_manager.h"
#include "led_manager.h"
#include "alert_manager.h"

String savedSSID = "";
String savedPassword = "";
String savedAlertApiUrl = DEFAULT_ALERT_API_URL;
String savedAlertApiToken = DEFAULT_ALERT_API_TOKEN;

bool isAPMode = false;
unsigned long apStartTime = 0;

bool touchButtonPressed = false;
unsigned long lastTouchTime = 0;

int currentDisplayMode = DISPLAY_MODE_TIME;
const int MAX_DISPLAY_MODES = DISPLAY_MODES_COUNT;

ESP8266WebServer server(WEB_PORT);
ESP8266HTTPUpdateServer httpUpdater;

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_SERVER, UTC_OFFSET_SECONDS,
NTP_UPDATE_INTERVAL_MS);

Adafruit_NeoPixel pixels(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
int ledBrightness = LED_BRIGHTNESS_DEFAULT;
int noAlertDisplayMode = NO_ALERT_DISPLAY_MODE_DEFAULT;

int finalApiIndexToLedStripIndexMap[27];
unsigned long lastAlertFetchTime = 0;
unsigned long lastHeapLogTime = 0;

void initializeApiToLedMapping()
{
  for (int i = 0; i < 27; ++i)
  {
    finalApiIndexToLedStripIndexMap[i] = -1;
  }
  for (int api_idx = 0; api_idx < 27; ++api_idx)
  {
    const char *api_oblast_name = API_OBLAST_NAMES[api_idx];
```

```
    bool mapped = false;
    for (int physical_map_idx = 0; physical_map_idx < 29;
++physical_map_idx)
    {
      if (strcmp(api_oblast_name,
PHYSICAL_LED_REGION_NAMES[physical_map_idx]) == 0)
      {
        finalApiIndexToLedStripIndexMap[api_idx] =
PHYSICAL_LED_STRIP_INDICES[physical_map_idx];
        mapped = true;
        break;
      }
    }
    if (!mapped && api_idx != 18)
    {
      Serial.printf_P(PSTR("Увага: API регіон '%s' (API індекс %d)
не вдалося зіставити з LED.\n"), api_oblast_name, api_idx);
    }
  }
}

void setup()
{
  Serial.begin(115200);
  Serial.println(F("\nЗавантаження AlertMap..."));

  pinMode(TOUCH_BUTTON_PIN, INPUT);
  DisplayManager::init();

  EEPROMManager::init();
  EEPROMManager::loadWiFiCredentials(savedSSID, savedPassword);
  EEPROMManager::loadApiConfig(savedAlertApiUrl,
savedAlertApiToken);
  ledBrightness = EEPROMManager::loadLEDBrightness();
  noAlertDisplayMode = EEPROMManager::loadNoAlertDisplayMode();

  LEDManager::init();
  initializeApiToLedMapping();
  AlertManager::init();

  if (savedSSID.length() > 0 &&
WiFiManager::connectToWiFi(savedSSID, savedPassword))
  {
    isAPMode = false;
    MDNSManager::init();
    timeClient.begin();
    Serial.println(F("Підключено до WiFi. NTP клієнт запущено."));
  }
  else
  {
    Serial.println(F("Не вдалося підключитися до WiFi або немає
облікових даних. Запущено режим AP."));
```

```
    WiFiManager::startAccessPoint();
  }

  WebServerManager::init();
  Serial.println(F("Веб-сервер запущено."));
}

void handleTouchButton()
{
  bool currentButtonState = digitalRead(TOUCH_BUTTON_PIN);
  unsigned long currentTime = millis();

  if (currentButtonState == HIGH && !touchButtonPressed &&
(currentTime - lastTouchTime > 200))
  {
    touchButtonPressed = true;
    lastTouchTime = currentTime;

    if (!isAPMode)
    {
      currentDisplayMode = (currentDisplayMode + 1) %
MAX_DISPLAY_MODES;
    }
    else
    {
      Serial.println(F("Кнопка натиснута в режимі AP. Очищення
облікових даних WiFi та перезавантаження."));
      EEPROMManager::clearWiFiCredentials();
      EEPROMManager::clearApiConfig();
      delay(1000);
      ESP.restart();
    }
  }
  else if (currentButtonState == LOW && touchButtonPressed)
  {
    touchButtonPressed = false;
  }
}

void loop()
{
  unsigned long currentTime = millis();

  if (currentTime - lastHeapLogTime >= 60000)
  {
    Serial.printf_P(PSTR("Loop: Вільна пам'ять: %u. Час роботи: %lu
ms\n"), ESP.getFreeHeap(), currentTime);
    lastHeapLogTime = currentTime;
  }

  handleTouchButton();
  WebServerManager::handleClient();
```

```
  if (!isAPMode)
  {
    MDNSManager::update();
  }

  DisplayManager::update();

  if (WiFiManager::isConnected() && !isAPMode)
  {
    if (currentTime - lastAlertFetchTime >= ALERT_UPDATE_INTERVAL_MS
|| lastAlertFetchTime == 0)
    {
      String newAlerts = AlertManager::fetchAlerts();
      if (newAlerts.length() == 27)
      {
        LEDManager::setAlertData(newAlerts);
      }
      else
      {
        LEDManager::setAlertData("");
      }
      lastAlertFetchTime = currentTime;
    }
  }

  LEDManager::update();
  WiFiManager::checkAPTimeout();

  if (!isAPMode && !WiFiManager::isConnected())
  {
    Serial.println(F("WiFi з'єднання втрачено. Перезавантаження
через 3 секунди..."));
    delay(3000);
    ESP.restart();
  }

  delay(50);
}
#ifndef ALERT_MANAGER_H
#define ALERT_MANAGER_H

#include "config.h"

class AlertManager
{
public:
    static void init()
    {
    }

    static String fetchAlerts()
```

```
    {

        std::unique_ptr<BearSSL::WiFiClientSecure> client(new
BearSSL::WiFiClientSecure);
        if (!client)
        {
            Serial.println(F("AlertManager: Помилка виділення
WiFiClientSecure!"));
            return "";
        }
        client->setInsecure();
        client->setBufferSizes(512, 512);

        HTTPClient https;
        String fullUrl = savedAlertApiUrl + "?token=" +
savedAlertApiToken;
        String alertStr = "";


        if (https.begin(*client, fullUrl))
        {
            int httpCode = https.GET();

            if (httpCode > 0)
            {
                if (httpCode == HTTP_CODE_OK || httpCode ==
HTTP_CODE_MOVED_PERMANENTLY)
                {
                    String payload = https.getString();

                    DynamicJsonDocument doc(512);
                    DeserializationError error =
deserializeJson(doc, payload);


                    if (!error)
                    {
                        if (doc.is<JsonObject>() &&
doc.as<JsonObject>().containsKey("alerts_string"))
                        {
                            String extractedString =
doc["alerts_string"].as<String>();
                            if (extractedString.length() == 27)
                            {
                                alertStr = extractedString;
                            }
                        }
                        else if (doc.is<JsonVariant>() &&
doc.as<JsonVariant>().is<const char *>())
                        {
                            String extractedString =
doc.as<String>();
```

```
                            if (extractedString.length() == 27)
                            {
                                alertStr = extractedString;
                            }
                        }
                    }
                    else
                    {
                        if (payload.startsWith("\"") &&
payload.endsWith("\""))
                        {
                            String rawString = payload.substring(1,
payload.length() - 1);
                            if (rawString.length() == 27)
                            {
                                alertStr = rawString;
                            }
                        }
                        else if (payload.length() == 27)
                            alertStr = payload;
                        }
                        if (alertStr.isEmpty())
                            Serial.println(F("AlertManager: Помилка
розбору JSON або невідомий формат відповіді."));
                            Serial.print(F("Помилка deserializeJson:
"));
                            Serial.println(error.c_str());
                        }
                    }
                }
                else
                {
                    Serial.printf_P(PSTR("AlertManager: HTTP GET
запит не вдався, неочікуваний HTTP код: %d\n"), httpCode);
                }
            }
            else
            {
                Serial.printf_P(PSTR("AlertManager: HTTP GET запит
не вдався, помилка: %s\n"), https.errorToString(httpCode).c_str());
            }
            https.end();
        }
        else
        {
            Serial.printf_P(PSTR("AlertManager: Не вдалося
підключитися до %s (https.begin не вдався).\n"), fullUrl.c_str());
        }

        if (alertStr.length() != 27 && !alertStr.isEmpty())
        {
```

```
            Serial.printf_P(PSTR("AlertManager: Попередження -
отримано рядок тривог невірної довжини: %d\n"), alertStr.length());
        }
        else if (alertStr.isEmpty() && WiFi.status() ==
WL_CONNECTED)
        {
            Serial.println(F("AlertManager: Не вдалося отримати дані
тривог."));
        }
        return alertStr;
    }
};

#endif
```

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

РЕЦЕНЗІЯ
на кваліфікаційну роботу

випускника спеціальності: 123 «Комп'ютерна інженерія»

відділення: комп'ютерної та програмної інженерії

циклова комісія: комп'ютерних систем та мереж

Іван САГАЙДАК
(ім'я, прізвище)

1. Обрана тема кваліфікаційної роботи, «Проектування фізичної моделі карти повітряних тривог України на базі мікроконтролера ESP8266», є актуальною.

2. Кваліфікаційна робота відповідає темі, затвердженій наказом.

3. Завдання на виконання кваліфікаційної роботи виконано у повному обсязі у встановлений термін.

4. В результаті виконання кваліфікаційної роботи було виконано проектування та створення фізичної моделі карти України, на якій візуально відображаються повітряні тривоги в певних областях та регіонах.

5. Якість виконання пояснювальної записки та ілюстративного (графічного) матеріалу відповідає вимогам Державних стандартів.

6. В кваліфікаційній роботі зроблений акцент на розробці програмного забезпечення для мікроконтролера ESP8266 і на отриманні оперативних даних про повітряні тривоги.

7. Кваліфікаційна робота заслуговує оцінку «відмінно», а випускник присвоєння кваліфікації фахівця освітньо-професійного ступеня «Фаховий молодший бакалавр» спеціальності 123 «Комп'ютерна інженерія».

Рецензент _____
(науковий ступінь, посада)

«____»_____ 2025 р. _____ Марія КИСЛОВА
(підпис) (ім'я, прізвище)

З рецензією ознайомлений _____ Іван САГАЙДАК
(підпис) (ім'я, прізвище)

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

ВІДГУК
керівника кваліфікаційної роботи

випускника спеціальності: 123 «Комп'ютерна інженерія»

відділення: комп'ютерної та програмної інженерії

циклова комісія: комп'ютерних систем та мереж

Іван САГАЙДАК
(ім'я, прізвище)

19. Кваліфікаційна робота на тему «Проектування фізичної моделі карти повітряних тривог України на базі мікроконтролера ESP8266» виконана в ініціативному порядку.

20. Метою кваліфікаційної роботи є створення засобу, який зручно та своєчасно може інформувати про наявність повітряної тривоги в певним областях та окремих регіонах.

21. Кваліфікаційна робота відповідає темі, затвердженій наказом начальника коледжу та виконана здобувачем освіти самостійно.

22. Здобувач освіти показав високі вміння роботи з літературними джерелами, аналіз теоретичного та практичного матеріалу, приймання обґрунтованих рішень, застосовування сучасних комп'ютерних інформаційних технологій.

23. Здобувач освіти показав високий рівень дотримання вимог державних стандартів при виконанні кваліфікаційної роботи в цілому та оформленні пояснювальної записки.

24. Рівень виконаної кваліфікаційної роботи заслуговує оцінку «відмінно», відповідає набутих випускником знань, умінь та навичок, вимогам освітньої характеристики фахівця і можливість присвоєння йому кваліфікації фахівця освітньо-професійного ступеня «Фаховий молодший бакалавр» спеціальності 123 «Комп'ютерна інженерія».

Керівник кваліфікаційної роботи

«09» 06 2025 р. _____ Роман МІНЕНКО
(підпис) (ім'я, прізвище)