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)
(підпис) *Ірина КРАВЧУК* (ім'я, ПРІЗВИЩЕ)
« 10 » 06 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-ПРОФЕСІЙНОГО СТУПЕНЯ
ФАХОВИЙ МОЛОДШИЙ БАКАЛАВР

Тема: *Розумний годинник на базі ESP-01 з функцією відображення прогнозу погоди*

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

Здобувач освіти (підпис) *Михайло ДЕРЕВ'ЯГА* (ім'я, ПРІЗВИЩЕ)

Керівник роботи (підпис) *Тетяна РУБАН* (ім'я, ПРІЗВИЩЕ)

Консультант з оформлення пояснювальної записки (підпис) *Оксана ОСАДЧА* (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)
_____ *Ірина КРАВЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)
« *10* » *03* 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

ДЕРЕВ'ЯГИ Михайла Романовича
(прізвище, ім'я, по батькові)

1.Тема роботи *Розумний годинник на базі ESP-01 з функцією відображення прогнозу погоди*

Керівник роботи *Рубан Тетяна Миколаївна, викладач першої категорії*
(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 _____ по _____

3. Вихідні дані до роботи *Розробка макету розумний годинник*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *обґрунтування вибору архітектури розумного годинника, розробка функціональної схеми пристрою, підключення та конфігурування OLED-дисплея, реалізація підключення до Wi-Fi, отримання та обробка прогнозу погоди через API, програмування мікроконтролера ESP-01 в середовищі Arduino IDE.*

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*	*22.03.2025-24.03.2025*	*виконано*
2	*Підбір та вивчення науково-технічної літератури за темою кваліфікаційної роботи*	*25.03.2025-31.03.2025*	*виконано*
3	*Обґрунтування вибору програмних засобів*	*01.04.2025-06.04.2025*	*виконано*
4	*Опис компонентів. Обґрунтування їх вибору.*	*07.04.2025-10.04.2025*	*виконано*
5	*Розробка програмного забезпечення*	*11.04.2025-25.04.2025*	*виконано*
6	*Дослідження ефективності реалізованих методів.*	*26.04.2025-03.05.2025*	*виконано*
7	*Написання пояснювальної записки*	*04.05.2025-22.05.2025*	*виконано*
8	*Перевірка на плагіат пояснювальної записки*	*25.05.2025-01.06.2025*	*виконано*
9	*Попередній захист кваліфікаційної роботи*	*02.06.2025-07.06.2025*	*виконано*
10	*Захист кваліфікаційної роботи*		*виконано*

Здобувач освіти _____ (підпис) *Михайло ДЕРЕВ'ЯГА* (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) *Тетяна РУБАН* (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «*Розумний годинник на базі ESP-01 з функцією відображення прогнозу погоди*»: 64 сторінки основного тексту, 27 рисунків, 1 таблиця, 15 використаних джерел, 1 додаток.

ESP-01, РОЗУМНИЙ ГОДИННИК, *IoT*, МІКРОКОНТРОЛЕР, *OLED*-ДИСПЛЕЙ, ПРОГНОЗ ПОГОДИ, *WEATHERAPI*, АВТОМАТИЗОВАНА СИСТЕМА, *Wi-Fi*, ВЕБ-ІНТЕРФЕЙС

Метою кваліфікаційної роботи є проєктування та реалізація прототипу розумного годинника на базі мікроконтролера *ESP*-01 з функцією отримання й відображення прогнозу погоди в режимі реального часу.

У процесі виконання роботи було проаналізовано предметну область, досліджено апаратні та програмні засоби для створення *IoT*-пристроїв, обрано погодний *API* (*WeatherAPI*) для інтеграції прогнозу погоди, реалізовано алгоритм обробки метеоданих та розроблено веб-інтерфейс для налаштування пристрою.

У третьому розділі описано структуру програмної реалізації, взаємодію *ESP*-01 з дисплеєм і мережею, способи отримання даних через інтернет, а також проведено тестування пристрою в реальних умовах.

У результаті виконання кваліфікаційної роботи було створено повнофункціональний прототип розумного годинника, здатного виводити актуальну інформацію про погоду, час і додаткові параметри навколишнього середовища. Пристрій має компактні розміри, базується на енергоефективних компонентах та демонструє потенціал для подальшого розвитку в рамках систем розумного дому та персональних *IoT*-рішень.

Додано примітку [OO1]: Перевірте правильність наведених даних

ЗМІСТ

Додано примітку [OO2]: Перевірте нумерацію сторінок

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

ESP-01 – компактний *Wi-Fi*-модуль на базі мікроконтролера *ESP*8266, який широко використовується в *IoT*-проєктах для підключення до Інтернету та обробки даних.

IoT (Internet of Things) – Інтернет речей. Концепція, згідно з якою фізичні об'єкти мають можливість передавати дані через мережу без участі людини.

OLED (*Organic Light-Emitting Diode*) – тип дисплею з високою контрастністю та низьким енергоспоживанням, використовується для виводу текстової та графічної інформації.

Wi-Fi – бездротова технологія передачі даних, яка дозволяє пристроям підключатися до локальної мережі або Інтернету.

API (*Application Programming Interface*) – інтерфейс програмного застосунку, який дозволяє пристрою отримувати зовнішні дані або послуги, наприклад, прогноз погоди.

WeatherAPI – вебсервіс для отримання поточної метеорологічної інформації: температура, вологість, напрямок і швидкість вітру, тиск, якість повітря та інше.

NTP (*Network Time Protocol*) – протокол синхронізації годинника мікроконтролера з інтернет-серверами точного часу.

JSON (*JavaScript Object Notation*) – формат зберігання та передачі структурованих даних у вигляді тексту, який зручно обробляти програмно.

HTTP-запит – запит, що відправляється через Інтернет для отримання або передачі даних з віддаленого сервера.

IDE (*Integrated Development Environment*) – програмне середовище для розробки, яке об'єднує редактор коду, компілятор та інструменти відлагодження (у цьому проєкті використано *Arduino IDE*).

*SSD*1306 – контролер графічного *OLED*-дисплея з роздільною здатністю 128*x*64 пікселів, який підтримує *I2C* та *SPI* інтерфейси.

ВСТУП

У сучасному світі стрімко зростає популярність пристроїв, які забезпечують комфорт, автоматизацію та інформативність повсякденного життя. Одним із найбільш динамічних напрямів розвитку є сфера інтернету речей (*IoT*), у якій провідне місце займають розумні пристрої, здатні збирати, обробляти та виводити дані з інтернет-джерел. Особливу увагу користувачів і розробників привертають малі автономні пристрої з широким функціоналом — серед них розумні годинники, які поєднують компактність, портативність і доступ до важливої інформації в режимі реального часу.

Розумний годинник — це електронний пристрій, що виконує функції звичайного годинника, доповнені можливістю відображення додаткових даних, таких як прогноз погоди, якість повітря, вологість, швидкість і напрям вітру, атмосферний тиск тощо. Завдяки підключенню до мережі *Wi-Fi* та використанню погодних *API*, пристрій може регулярно оновлювати інформацію, відображаючи її на вбудованому дисплеї в зручному для користувача вигляді. Така система має велике значення для побутового застосування, коли користувач хоче миттєво отримувати метеодані без потреби користуватися смартфоном або комп’ютером.

У межах цієї роботи передбачається розробити прототип розумного годинника з можливістю відображення прогнозу погоди на основі мікроконтролера *ESP*-01. У пристрої використовуються *OLED*-дисплей, модуль *Wi-Fi*, а також *API*-сервіс для отримання актуальних погодних даних (наприклад, *WeatherAPI*). Годинник демонструє температуру повітря, вологість, тиск, напрям і швидкість вітру, поточну дату та час, а також текстову та графічну індикацію стану погоди. Завдяки компактному форм-фактору пристрій можна використовувати вдома, в офісі або в навчальному закладі.

Актуальність теми обумовлена зростанням попиту на мініатюрні, енергоефективні та інформативні рішення, що працюють на базі *IoT*-

платформ. Реалізація подібного пристрою демонструє потенціал мікроконтролерів *ESP*-серії у побудові бюджетних, але функціональних систем з доступом до інтернету.

Метою даної роботи є розробка прототипу розумного годинника з функцією відображення прогнозу погоди, використовуючи мікроконтролер *ESP*-01, *OLED*-дисплей та інтеграцію з погодним *API*.

З мети роботи випливають такі завдання:

- проаналізувати предметну область та існуючі рішення;
- дослідити технічні характеристики платформи *ESP*-01 і *OLED*-дисплеїв;
- обрати та налаштувати джерело погодних даних (*API*);
- розробити алгоритм отримання, обробки та виводу метеоданих;
- реалізувати веб-інтерфейс для налаштування параметрів пристрою;
- протестувати пристрій у реальних умовах та оцінити його ефективність.

Об'єктом дослідження є мікроконтролер *ESP*-01 як основа для побудови розумного *IoT*-пристрою.

Предмет дослідження – є апаратні та програмні рішення для реалізації функціонального розумного годинника з можливістю відображення прогнозу погоди, зокрема способи отримання та обробки метеоданих, методи їх виводу на екран і стабільної роботи з мережею.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна характеристика розумного годинника

У сучасному цифровому суспільстві розумні технології дедалі глибше інтегруються в повсякденне життя людини. Одним із прикладів такої інтеграції є розумний годинник — компактний носимий пристрій, який, зберігаючи базові функції звичайного хронометра, водночас виконує роль персонального асистента, фітнес-трекера, інтерфейсу для повідомлень і, в деяких випадках, навіть елемента керування іншими пристроями в екосистемі інтернету речей.

Основною особливістю розумного годинника є його здатність обробляти дані в режимі реального часу, взаємодіяти з іншими цифровими пристроями та отримувати доступ до мережевих сервісів. Його конструкція зазвичай включає мікропроцесор або мікроконтролер, модулі зв’язку (таких як *Wi-Fi* або *Bluetooth*), сенсори для вимірювання фізичних параметрів (наприклад, акселерометр, пульсометр, гіроскоп), а також дисплей для виведення даних користувачеві. Усе це поєднується в компактному корпусі, зручному для постійного носіння на руці.

На відміну від звичайних годинників, які виконують лише одну функцію показ часу, розумні годинники перетворюються на багатофункціональні пристрої. Вони здатні не лише інформувати про поточну дату чи погоду, а й надавати аналітику щодо фізичної активності користувача, контролювати якість сну, синхронізуватися зі смартфонами, дозволяючи приймати дзвінки, читати повідомлення, керувати відтворенням музики тощо. Деякі моделі навіть підтримують роботу з голосовими асистентами, картами, оплатою товарів за допомогою безконтактних технологій.

Розвиток технологій дозволив сформувати цілий спектр моделей розумних годинників — від найпростіших фітнес-браслетів до

високотехнологічних пристроїв з мобільною операційною системою. Паралельно з комерційними рішеннями активно розвивається сегмент відкритих, саморобних пристроїв, створених ентузіастами на базі мікроконтролерів і платформ вільного програмного забезпечення. Саме такий підхід є особливо привабливим для дослідницьких та навчальних проєктів, оскільки дозволяє гнучко адаптувати функціональність пристрою до конкретних завдань, вивчати особливості мікроелектроніки та програмування в умовах реальної розробки.

Розумні годинники, будучи частиною ширшої концепції інтернету речей, здатні забезпечити користувачеві постійний зв'язок із цифровим середовищем. Наприклад, використовуючи бездротове з'єднання, вони можуть отримувати дані з мережі (такі як прогноз погоди), автоматично синхронізувати час через *NTP*-сервери, реагувати на події з інших пристроїв розумного дому або передавати дані до хмарних сервісів для подальшого аналізу. Це дає змогу говорити про розумний годинник як про вузлову ланку в особистій цифровій інфраструктурі користувача.

Наявність розумного годинника створює нові сценарії взаємодії з інформацією. Наприклад, у ситуаціях, коли користувач не має змоги дістати смартфон, годинник дозволяє оперативно ознайомитися з важливими повідомленнями або погодними умовами, не втрачаючи при цьому часу. Це, у свою чергу, підвищує комфорт і ефективність повсякденної діяльності.

Підсумовуючи викладене, можна стверджувати, що розумний годинник — це не просто зручний аксесуар, а багатофункціональний інтелектуальний пристрій, який забезпечує новий рівень взаємодії людини з інформаційними системами. Його універсальність, мобільність, гнучкість налаштувань та здатність до автономної роботи відкривають широкі перспективи для його подальшого розвитку, а також для досліджень у межах академічних і прикладних проєктів, що базуються на принципах *IoT*.

1.2 Огляд апаратного забезпечення для *IoT*-проєктів

Інтернет речей (*IoT - Internet of Things*) представляє собою концепцію об'єднання фізичних об'єктів у мережу через вбудовані технології для взаємодії між собою або з зовнішнім середовищем. Для реалізації *IoT*-проєктів необхідне спеціалізоване апаратне забезпечення, яке забезпечує збір, обробку та передачу даних. Вибір правильного апаратного забезпечення є критично важливим фактором успіху будь-якого *IoT*-проєкту.

Мікроконтролери для *IoT*-рішень

Arduino платформа залишається однією з найпопулярніших для прототипування та розробки *IoT*-пристроїв завдяки своїй простоті використання та великій спільноті розробників. *Arduino Uno R*3 базується на мікроконтролері *ATmega*328*P* з робочою частотою 16 МГц та включає 32 КБ *Flash* пам'яті, 2 КБ *SRAM* та 1 КБ *EEPROM*. Платформа пропонує 14 цифрових входів/виходів, 6 з яких підтримують ШІМ, та 6 аналогових входів. Робоча напруга складає 5В, а вартість коливається в межах 25-30 доларів США.

Компактніша версія *Arduino Nano* базується на тому ж *ATmega*328*P*, але має менші розміри 45×18 мм при збереженні основної функціональності. *Arduino Pro Mini* представляє найбільш компактну версію розміром 33×18 мм, проте не має вбудованого *USB*-інтерфейсу та потребує зовнішнього програматора. Головним недоліком платформи *Arduino* для *IoT*-застосувань є відсутність вбудованих засобів мережевого з'єднання, що потребує додаткових модулів.

*ESP*8266 сімейство мікроконтролерів революціонізувало сферу *IoT* завдяки вбудованому *Wi-Fi* модулю та надзвичайно низькій вартості. *NodeMCU v*3 базується на мікроконтролері *ESP*8266*EX* з частотою процесора 80/160 МГц та включає 4 МБ *Flash* пам'яті і 96 КБ *SRAM*. Платформа підтримує *Wi-Fi* стандарту *IEEE* 802.11 *b*/*g*/*n* та пропонує 17 *GPIO* пінів, один 10-бітний АЦП канал та програмно керовані ШІМ виходи. Розміри плати складають 58×31 мм, а вартість не перевищує 12 доларів США.

*Wemos D*1 *Mini* представляє компактну версію на базі *ESP*8266 розміром 34×26 мм з 11 цифровими входами/виходами та *Micro USB* для програмування. *ESP*-01 та *ESP*-01*S* є найбільш компактними модулями *ESP*8266 розміром 25×15 мм з 4 доступними *GPIO* пінами. Попри обмежену кількість виводів, ці модулі ідеально підходять для простих *IoT*-проєктів завдяки надзвичайно низькій вартості 2-4 долари США.

*ESP*32 представляє наступне покоління мікроконтролерів з розширеними можливостями. *ESP*32 *DevKit V*1 оснащений двоядерним процесором *Xtensa LX*6 з частотою 240 МГц, до 16 МБ *Flash* пам'яті та 520 КБ *SRAM*. Платформа підтримує *Wi-Fi IEEE* 802.11 *b*/*g*/*n* та *Bluetooth v*4.2 *BR*/*EDR* і *BLE*, пропонує 36 *GPIO* пінів, два 12-бітних АЦП, два 8-бітних ЦАП та множинні інтерфейси *UART*, *SPI*, *I2C* та *I2S*. *ESP*32-*CAM* включає модуль камери *OV*2640 та слот для *microSD* карт, що робить його ідеальним для проєктів відеоспостереження.

Таблиця 1.1 – Список використаних компонентів

Найменування	**Пакет/Маркування**	**Кількість**
DIP E-capacitor 10uF	*C1, C2*	2
0.96 inch OLED display module	*JP1*	1
Horizontal button	*KEY1*	1
WIFI module	*P1*	1
USB interface	*P2*	1
AMS1117-3.3V voltage regulator chip	*U1*	1
Bent foot single row 4P socket	*JP1*	1
Double row 4P socket	*P1*	1
USB power cable		1
Shell		1
Screw bag		1
PCB		1

Raspberry Pi Zero W представляє повноцінний комп'ютер з процесором *ARM*11 1 ГГц, 512 МБ *RAM*, вбудованими *Wi-Fi* та *Bluetooth* модулями. Платформа підтримує повноцінну операційну систему *Linux* та мови програмування *Python*, *C++*, *Java*, що робить її придатною для складних обчислювальних завдань. Вартість складає 15-20 доларів США при значно вищих можливостях порівняно з мікроконтролерами. Список усіх використаних компонентів представленно у таблиці 1.1

1.3 Прогноз погоди: доступні *API* та методи інтеграції

Сучасні смарт-годинники активно використовують інтернет-сервіси для отримання актуальної інформації про погоду. Для реалізації функції прогнозу погоди в розумному годиннику на базі *ESP*01-*S* необхідно провести аналіз доступних *API* та методів їх інтеграції.

OpenWeatherMap API є одним із найпопулярніших безкоштовних сервісів для отримання погодних даних. Сервіс надає *REST API* з можливістю отримання поточної погоди, прогнозу на 5 днів та історичних даних. Для *ESP*8266 особливо важливою є простота інтеграції - *API* повертає дані у форматі *JSON*, що легко парситься на мікроконтролері. Безкоштовний план дозволяє до 1000 запитів на день, що цілком достатньо для персонального пристрою. *API* підтримує запити за географічними координатами, назвою міста або *ZIP*-кодом.

WeatherAPI пропонує більш *generous* безкоштовний план - до 1 мільйона запитів на місяць. Сервіс надає детальну інформацію про поточну погоду, прогноз до 10 днів, астрономічні дані (схід/захід сонця, фази місяця). *JSON* відповіді добре структуровані та містять всю необхідну інформацію для відображення на дисплеї годинника. *API* також підтримує отримання погодних попереджень та індексів (*UV*, якість повітря).

AccuWeather API відомий своєю точністю прогнозів, особливо для локальних умов. Безкоштовний план обмежений 50 запитами на день, але

надає дуже детальну інформацію включаючи відчувану температуру, вологість, швидкість вітру, видимість. *API* має окремі ендпоінти для поточних умов, годинного прогнозу та 5-денного прогнозу.

*ESP*01-*S* підтримує *HTTP*/*HTTPS* запити через *WiFi* з'єднання, що робить інтеграцію з погодними *API* достатньо простою. Основні методи реалізації включають використання бібліотек *ESP*8266*Wi-Fi* та *ESP*8266*HTTPClient* для *Arduino IDE*. Типовий алгоритм роботи передбачає підключення до *WiFi* мережі, формування *HTTP GET* запиту з *API* ключем та параметрами локації, отримання *JSON* відповіді та парсинг необхідних даних.

Важливим аспектом є оптимізація частоти запитів для економії енергії та дотримання лімітів *API*. Рекомендується кешувати погодні дані та оновлювати їх не частіше ніж раз на 10-15 хвилин для поточної погоди та раз на годину для прогнозу. Також необхідно передбачити обробку помилок мережі та недоступності *API* сервісу.

Для парсингу *JSON* на *ESP*01-*S* ефективно використовувати бібліотеку *ArduinoJson*, яка оптимізована для роботи з обмеженою пам'яттю мікроконтролера. При проектуванні важливо враховувати, що *ESP*01-*S* має лише 1МБ *flash* пам'яті, тому код повинен бути оптимізованим.

Безпека та надійність

При роботі з зовнішніми *API* критично важливою є безпека зберігання *API* ключів та обробка *SSL*/*TLS* з'єднань. *ESP*01-*S* підтримує *HTTPS*, але це потребує додаткових ресурсів пам'яті. Альтернативою може бути використання *HTTP* з додатковою аутентифікацією або проксі-сервера.

Для забезпечення надійності роботи необхідно реалізувати систему резервних *API* - якщо основний сервіс недоступний, система може перемикатися на альтернативний. Також важливим є локальне кешування останніх отриманих даних для відображення інформації навіть при тимчасовій втраті інтернет-з'єднання.

1.4 Аналіз існуючих рішень для реалізації розумного годинника

Сучасний ринок смарт-годинників представлений широким спектром пристроїв від бюджетних моделей до преміум-сегменту. *Apple Watch Series* 9 представлено на рисунку 1.2, представляє еталон функціональності з власною операційною системою *watchOS*, підтримкою додатків, *GPS*, *NFC* для безконтактних платежів, моніторингом здоров'я включаючи ЕКГ та рівень кисню в крові. Дисплей *Always-On Retina* забезпечує відмінну читабельність при будь-якому освітленні. Водонепроникність *IPX*8 та автономність до 18 годин роблять його ідеальним для щоденного використання.

Рисунок 1.2 – *Apple Watch Series* 9

Samsung Galaxy Watch 6 працює на операційній системі *Wear OS* та пропонує аналогічну функціональність з акцентом на інтеграцію з *Android* пристроями. Особливістю є обертовий безель для навігації, більш тривала батарея (до 40 годин) та розширені можливості фітнес-трекінгу. Підтримує *eSIM* для незалежної роботи від смартфона (рисунок 1.3).

Рисунок 1.3 – *Samsung Galaxy Watch*6

Xiaomi Mi Band 8 представляє бюджетний сегмент з базовою *Smart*-функціональністю. Незважаючи на низьку ціну, пристрій підтримує моніторинг активності, пульсометр, відображення повідомлень та має автономність до 16 днів. *AMOLED* дисплей 1.64" забезпечує гарну читабельність (рисунок 1.4).

Рисунок 1.4- Xiaomi Mi Band 8

LILYGO T-Watch 2020 - це повнофункціональна платформа для розробки смарт-годинника на базі *ESP*32. Включає 1.54" *TFT* дисплей, *WiFi*, *Bluetooth*, *GPS*, акселерометр, мікрофон та динамік. Підтримує *Arduino IDE* та *ESP-IDF*. Відкритий дизайн дозволяє повністю кастомізувати функціональність. Активна спільнота розробників створила множину прошивок від простих годинників до повноцінних *smartwatch* з підтримкою додатків. Представлено на рисунку 1.5

Рисунок 1.5 - *LILYGO T-Watch 2020*

Watchy by SQFMI - *E-ink* смарт-годинник з відкритим кодом на базі *ESP*32. Унікальність проекту в використанні електронних чорнил, що забезпечує надзвичайно низьке енергоспоживання - батарея тримає тижнями. 1.54" *E-paper* дисплей ідеально читається при яскравому сонячному світлі. Програмується через *Arduino IDE*, має модульну архітектуру та активну спільноту розробників. Представлено на рисунку 1.6

Рисунок 1.6- *Watchy by SQFMI*

*ESP*32-C3 *Watch* - мінімалістичний проект смарт-годинника з акцентом на довгу автономність. Використовує *ESP*32-*C*3 з *RISC-V* архітектурою, 0.96" *OLED* дисплей, *RTC* модуль та *Li-Po* батарею. Підтримує базові функції: час, дата, будильник, секундомір. Повний код доступний на *GitHub* з детальними інструкціями збирання. Представлено на рисунку 1.7

Рисунок 1.7- *ESP*32-C3 *Watch*

*Arduino-based Smartwatch V*3 - популярний проект на *GitHub* з більш ніж 2000 зірок. Використовує *Arduino Nano*, 1.3" *OLED* дисплей, *RTC* модуль *DS*3231, *Li-Po* батарею та модуль *WiFi*. Функціональність включає відображення часу, будильник, секундомір, підключення до смартфона через *Bluetooth* для отримання повідомлень. Представлено на рисунку 1.8

Рисунок 1.8- *Arduino Smartwatch V*3

WatchX - open-source платформа для створення смарт-годинника з акцентом на навчання програмування. Включає *ATmega*328*P* мікроконтролер, 1.28" *TFT* дисплей, акселерометр, вібромотор, зумер. Підтримується графічне програмування через блоки та традиційне програмування на *C++*.

Порівняльний аналіз архітектурних рішень

Аналіз існуючих рішень показує декілька основних архітектурних підходів. Комерційні пристрої використовують потужні багатоядерні процесори (*Apple S*9, *Samsung Exynos W*930) з виділеною графікою для забезпечення плавної анімації та швидкого відгуку інтерфейсу. Операційні системи (*watchOS, Wear OS*) надають високорівневі *API* для розробки додатків.

DIY проекти зазвичай базуються на *ESP*32/*ESP*8266 або *Arduino*-сумісних мікроконтролерах. *ESP*32 пропонує оптимальне співвідношення продуктивності, функціональності та енергоспоживання з вбудованими *WiFi*/*Bluetooth* та достатньою *flash*/*RAM* пам'яттю для складних додатків.

Ключовими компонентами успішного смарт-годинника є якісний дисплей з хорошою читабельністю, ефективне управління живленням, надійне

бездротове з'єднання та зручний інтерфейс користувача. Відкриті проекти демонструють можливість створення функціональних пристроїв з бюджетом в 10-20 разів меншим за комерційні аналоги при збереженні основної функціональності.

РОЗДІЛ 2
ПРОЕКТУВАННЯ РОЗУМНОГО ГОДИННИКА З ВЕБ-ІНТЕРФЕЙСОМ

2.1 Вибір мікроконтролера *ESP*-01: характеристики, можливості, переваги

Мікроконтролер *ESP*-01 (представлено на рисунку 2.1) є одним з найпопулярніших та найдоступніших рішень для проектів Інтернету речей завдяки своїм компактним розмірам та вбудованому модулю *Wi-Fi*. Цей пристрій базується на чіпі *ESP*8266, який забезпечує потужну функціональність при мінімальному енергоспоживанні.

ESP-01 працює на частоті 80 МГц з можливістю розгону до 160 МГц, що цілком достатньо для обробки даних годинника та керування веб-інтерфейсом. Обсяг оперативної пам'яті становить 80 КБ, а флеш-пам'яті – 1 МБ, чого вистачає для зберігання програмного коду та базових налаштувань пристрою.

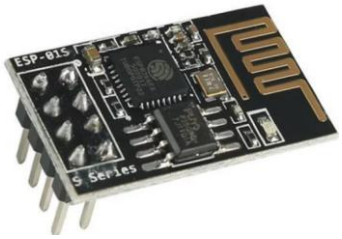

Рисунок 2.1 – *ESP*-01*S MODULE*

Особливо важливою перевагою *ESP*-01 є наявність вбудованого *Wi-Fi* модуля стандарту 802.11 *b*/*g*/*n*, який підтримує режими точки доступу та клієнта. Це дозволяє пристрою підключатися до існуючих мереж або створювати власну точку доступу для прямого підключення користувацьких пристроїв.

Мікроконтролер має обмежену кількість *GPIO* портів – лише 4 (представлено на рисунку 2.2), але цього достатньо для підключення дисплея та основних датчиків. Робоча напруга становить 3.3В, що потребує використання відповідних перетворювачів при живленні від стандартних джерел.

Значною перевагою *ESP*-01 є підтримка *Arduino IDE* та широка екосистема бібліотек, що значно спрощує процес розробки. Пристрій також підтримує *OTA* (*Over-The-Air*) оновлення, дозволяючи оновлювати прошивку через *Wi-Fi* без фізичного доступу до пристрою.

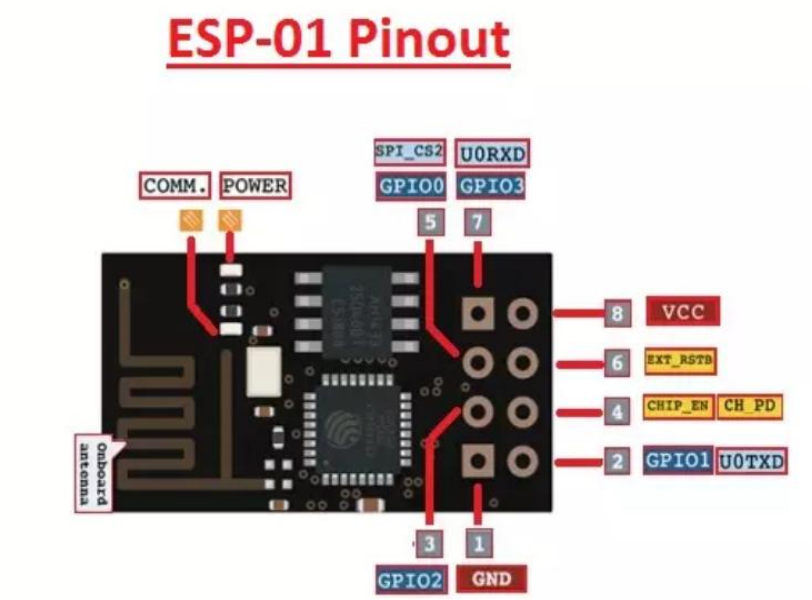

Рисунок 2.2 – Розпіновка *ESP*-01

2.2 Дисплеї для відображення інформації

Вибір дисплея для розумного годинника є критично важливим рішенням, що впливає на зручність використання, енергоспоживання та загальну функціональність пристрою. Для даного проекту розглядалися кілька варіантів дисплеїв, кожен з яких має свої особливості та область застосування.

OLED дисплеї на базі контролера *SSD*1306 (представлено на рисунку 2.3) представляють собою оптимальне рішення для портативних пристроїв. Такі екрани мають діагональ 0.96 дюйма з роздільною здатністю 128*x*64 пікселі, що забезпечує достатню деталізацію для відображення часу, дати та додаткової інформації. Головною перевагою *OLED* технології є відсутність підсвітлення – кожен піксель світиться самостійно, що забезпечує глибокий чорний колір та високий контраст навіть при яскравому освітленні.

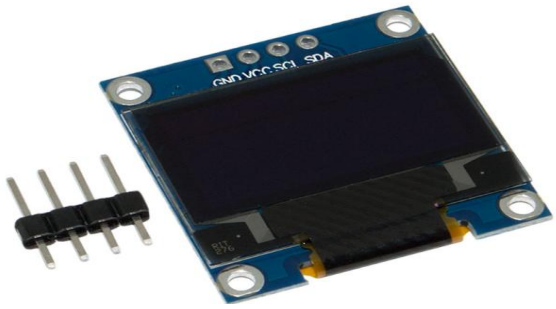

Рисунок 2.3 – *Oled display 0.96* з роздільною здатністю 128*x*64

Енергоспоживання *OLED* дисплеїв значно нижче порівняно з *LCD* екранами, особливо при відображенні темних зображень. Це критично важливо для годинника, який повинен працювати тривалий час від батареї. Інтерфейс підключення *I2C* дозволяє використовувати лише два *GPIO* порти мікроконтролера, залишаючи решту для інших компонентів (рисунок 2.4).

Альтернативним варіантом розглядався *TFT* дисплей з діагоналлю 1.8 дюйма та роздільною здатністю 160*x*128 пікселів. Такі екрани забезпечують кольорове відображення інформації та більшу площу для розміщення елементів інтерфейсу. Однак вони споживають значно більше енергії та потребують більшої кількості *GPIO* портів для підключення через *SPI* інтерфейс.

E-Paper дисплеї також розглядалися як варіант для годинника завдяки їх надзвичайно низькому енергоспоживанню. Такі екрани споживають енергію

лише під час оновлення зображення, що дозволяє досягти тижневої автономності. Проте повільна швидкість оновлення та обмежені можливості відображення динамічної інформації роблять їх менш придатними для інтерактивного годинника.

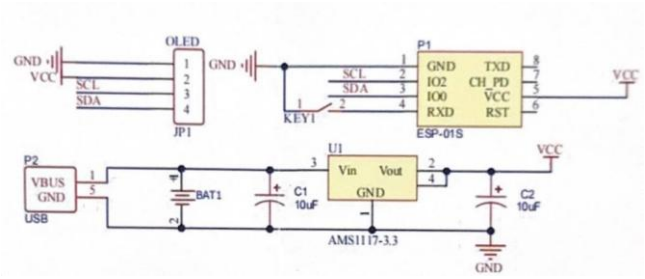

Рисунок 2.4 – Підключення обраного дисплея до *ESP*-01

2.3 Вибір способу передачі даних

Технологія *Wi-Fi* була обрана як основний спосіб передачі даних у розумному годиннику завдяки її універсальності, швидкості та широкій підтримці сучасними пристроями. *Wi-Fi* стандарту 802.11*n*, який підтримує *ESP*-01, забезпечує швидкість передачі даних до 150 Мбіт/с, що з великим запасом покриває потреби годинника для синхронізації часу, отримання погодних даних та взаємодії з веб-інтерфейсом представлено на рисунку 2.5.

Протокол *HTTP* використовується для комунікації між годинником та веб-браузером користувача. Це дозволяє створити зручний веб-інтерфейс, доступний з будь-якого пристрою в локальній мережі без необхідності встановлення спеціальних додатків. *ESP*-01 функціонує як веб-сервер, обробляючи запити від клієнтів та надсилаючи відповіді у форматі *HTML*, *CSS* та *JavaScript*.

Для отримання точного часу використовується протокол *NTP* (*Network Time Protocol*), який забезпечує синхронізацію з атомними годинниками через Інтернет з точністю до мілісекунд. Це критично важливо для годинника,

оскільки внутрішній генератор *ESP*-01 має природний дрейф, що призводить до накопичення похибки часу.

JSON формат застосовується для обміну структурованими даними між годинником та зовнішніми сервісами. Це особливо корисно при отриманні інформації про погоду, курси валют або інших динамічних даних. *JSON* забезпечує компактність передачі даних та легкість парсингу на стороні мікроконтролера.

Протокол *MQTT* розглядався як альтернативний варіант для інтеграції годинника в системи розумного дому. Цей протокол оптимізований для пристроїв з обмеженими ресурсами та забезпечує надійну доставку повідомлень навіть при нестабільному з'єднанні. Однак для базової функціональності годинника *HTTP* протокол виявився більш простим у реалізації.

Безпека передачі даних забезпечується використанням *WPA*2 шифрування на рівні *Wi-Fi* з'єднання. Для критично важливих налаштувань передбачена можливість використання *HTTPS* протоколу, хоча це потребує додаткових ресурсів мікроконтролера

2.4 Розробка принципової схеми та макету розумного годинника

Принципова схема розумного годинника (рисунок 2.5), відображає основні функціональні блоки та їх взаємозв'язки. Центральним елементом системи є мікроконтролер *ESP*-01, який виконує функції обробки даних, керування дисплеєм та забезпечення мережевої комунікації.

Блок живлення включає стабілізатор напруги на 3.3В, який перетворює напругу від батареї або зовнішнього адаптера до робочої напруги мікроконтролера.

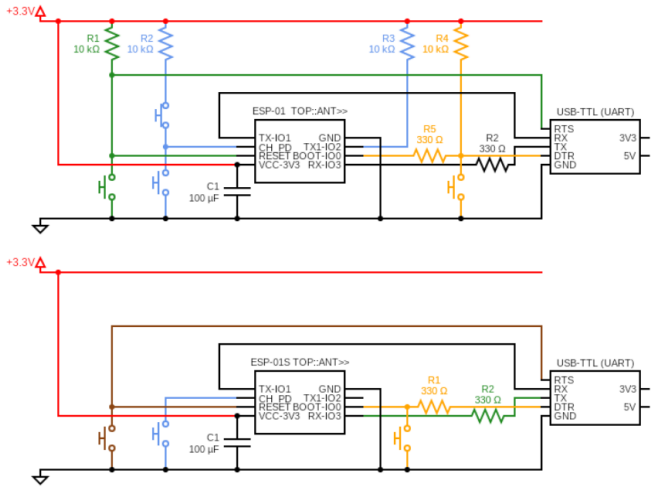

Рисунок 2.5 – Принципова схема розумного годинника

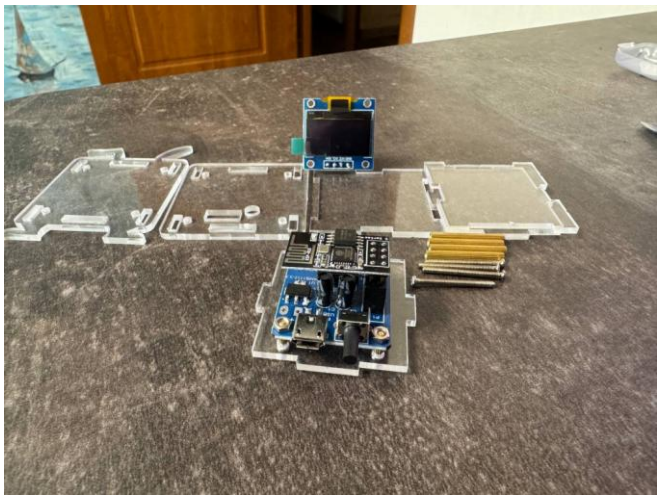

Рисунок 2.6 – Макет розумного годинника

Блок відображення інформації побудований на базі *OLED* дисплея з контролером *SSD*1306. Підключення здійснюється через *I2C* інтерфейс, що дозволяє використовувати лише два *GPIO* порти *ESP*-01. Резистори підтяжки на лініях *SDA* та *SCL* забезпечують стабільну роботу шини даних.

Блок користувацького інтерфейсу включає кнопки для навігації по меню та налаштування параметрів годинника. Використовуються тактові кнопки з підтяжними резисторами для запобігання помилкових спрацьовувань. Антидребізгові конденсатори забезпечують чистоту сигналів.

Принципова схема деталізує електричні з'єднання між компонентами. Мікроконтролер *ESP*-01 підключається до дисплея через *GPIO*0 (*SDA*) та *GPIO*2 (*SCL*). Кнопки підключені до *GPIO*1 та *GPIO*3 через струмообмежувальні резистори номіналом 10 кОм.

Схема живлення включає стабілізатор *AMS*1117-3.3 для перетворення напруги 5В у 3.3В. Фільтрувальні конденсатори на вході та виході стабілізатора забезпечують стабільність напруги живлення. *LED* індикатор показує стан живлення системи.

Для програмування *ESP*-01 передбачені контактні майданчики для підключення *USB-TTL* перетворювача. Це дозволяє завантажувати нові версії прошивки без демонтажу мікроконтролера з основної плати.

Розводка друкованої плати оптимізована для мінімальних розмірів при збереженні функціональності. Використовується двошарова плата з мінімальною шириною провідників 0.2 мм. Заземлюючі полігони забезпечують електромагнітну сумісність та зменшують перешкоди, готовий годинник представлено на рисунку 2.7

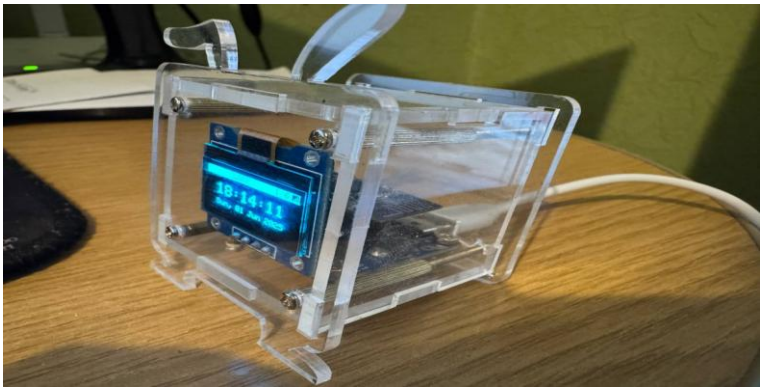

Рисунок 2.7 - Готова модель годинника

2.5 Вибір технологій для створення веб-інтерфейсу

У процесі розробки "розумного годинника на базі *ESP*-01 з функцією відображення прогнозу погоди" особливу увагу було приділено реалізації зручного та ефективного веб-інтерфейсу, який забезпечував би взаємодію користувача з пристроєм. Веб-інтерфейс повинен бути легким, швидким у завантаженні, не вимагати великих обчислювальних ресурсів з боку мікроконтролера та водночас забезпечувати всю необхідну функціональність. Для реалізації даного компонента було обрано відповідні мови програмування, бібліотеки, середовище розробки та допоміжні інструменти.

Мова програмування

Для створення веб-інтерфейсу було використано стандартний стек веб-технологій:

- *HTML* (*HyperText Markup Language*) – мова розмітки, що забезпечує структуру веб-сторінки. *HTML* використовується для побудови інтерфейсу, визначення блоків, кнопок, заголовків та інших елементів користувацького інтерфейсу.
- *CSS* (*Cascading Style Sheets*) – мова стилів, що використовується для оформлення зовнішнього вигляду елементів *HTML*. Завдяки *CSS* забезпечено адаптивність, привабливий вигляд інтерфейсу, зручність взаємодії на різних пристроях.
- *JavaScript* – мова сценаріїв, що виконується на стороні клієнта (в браузері). З її допомогою реалізовано динамічну взаємодію, обробку подій (наприклад, натискання кнопок, оновлення даних без перезавантаження сторінки), а також асинхронний обмін даними з пристроєм через *AJAX*-запити.

На стороні мікроконтролера (*ESP*-01) використано мову програмування *C++***,** яка є основною мовою для розробки прошивки у середовищі *Arduino IDE*. За допомогою цієї мови реалізовано веб-сервер, який обслуговує запити браузера та надсилає *HTML*-сторінку.

Середовище розробки

Основним середовищем розробки для всіх компонентів проекту виступила *Arduino IDE*, яке є зручним інструментом для програмування мікроконтролерів *ESP*. *Arduino IDE* підтримує завантаження бібліотек, компіляцію коду, прошивку пристрою, а також моніторинг послідовного порту для налагодження.

Додатково для редагування *HTML*/*CSS*/*JavaScript*-файлів було використано текстові редактори, такі як *Visual Studio Code*, які забезпечують підсвічування синтаксису, автодоповнення, перевірку помилок та попередній перегляд сторінок.

Бібліотеки

Для реалізації функціоналу веб-інтерфейсу були використані такі бібліотеки:

- *ESP*8266*WiFi.h* – стандартна бібліотека для роботи з *Wi-Fi* на мікроконтролерах *ESP*. Вона дозволяє налаштувати точку доступу, клієнтський режим або режим одночасної роботи (*AP + STA*).
- *ESPAsyncWebServer.h* – асинхронний веб-сервер для *ESP*8266, який дозволяє обслуговувати кілька одночасних запитів без блокування основного потоку виконання. Це особливо важливо для мікроконтролерів з обмеженими ресурсами.
- *LittleFS.h* – файлова система для збереження *HTML*/*CSS*/*JS*-файлів безпосередньо у флеш-пам’яті мікроконтролера. Це дозволяє відокремити прошивку від вмісту веб-інтерфейсу, спрощуючи оновлення дизайну або логіки без зміни основного коду.

- *Adafruit_SSD*1306 та *Adafruit_GFX* – бібліотеки для роботи з *OLED*-дисплеєм, не є частиною веб-інтерфейсу, але використовуються у синхронізації даних.
- Також у *JavaScript* при потребі може бути підключено сторонні бібліотеки, наприклад:
- *Chart.js* – для відображення графіків (температури, тиску тощо);
- *Bootstrap* – для реалізації адаптивної верстки без зайвих зусиль зі сторони розробника.

Формат зберігання та передача даних

Для забезпечення динамічності інтерфейсу, особливо при оновленні прогнозу погоди, використовуються *AJAX*-запити, які надсилаються з клієнтської частини до сервера, запущеного на *ESP*-01. Сервер відповідає даними у форматі ***JSON***, який легко обробляється на стороні *JavaScript*. Такий підхід дозволяє змінювати вміст сторінки без повного перезавантаження.

Крім того, у прошивці реалізовано маршрути:

- / – основна *HTML*-сторінка;
- /*weather* – обробка запиту на поточний прогноз;
- /*settings* – зміна параметрів (наприклад, місто для прогнозу, одиниці температури тощо).

Причини вибору технологій

Вибір зазначених технологій обумовлений наступними причинами:

- Продуктивність: *HTML*/*CSS*/*JS*-файли надзвичайно легкі та швидко завантажуються, що ідеально підходить для обмежених ресурсів *ESP*-01.
- Адаптивність: використання *JavaScript* забезпечує можливість реалізації інтерактивних елементів (наприклад, кнопок оновлення прогнозу, перемикачів режимів).
- Гнучкість: завдяки *LittleFS* веб-інтерфейс можна оновлювати без перепрошивки мікроконтролера.
- Зручність розробки: *Arduino IDE* та *Visual Studio Code* забезпечують комфортну розробку як прошивки, так і веб-частини.

Таким чином, використання мови *C++* на стороні мікроконтролера у поєднанні з *HTML, CSS, JavaScript* на клієнтській стороні, а також бібліотек *ESPAsyncWebServer* та *LittleFS* дозволило реалізувати ефективний, зручний і легкий веб-інтерфейс. Він забезпечує повноцінний контроль над пристроєм, відображення інформації та взаємодію з користувачем без потреби в зовнішніх серверах чи складних програмних рішеннях.

2.6. Створення та налаштування веб-інтерфейсу

Після вибору відповідних технологій розпочався процес безпосереднього створення та налаштування веб-інтерфейсу для взаємодії з розумним годинником на базі *ESP*-01. Основна мета полягала в тому, щоб надати користувачу інтуїтивно зрозумілу, швидкодіючу та функціональну веб-сторінку, яка дозволить переглядати інформацію про погоду, статус підключення до *Wi-Fi*, час, а також здійснювати базові налаштування.

Загальна структура інтерфейсу

- Веб-інтерфейс поділено на кілька логічних частин:
- Головна сторінка (/) – містить базову інформацію: поточний час, дату, погодні умови, статус *Wi-Fi*.
- Сторінка налаштувань (/*settings*) – дає можливість змінювати параметри, такі як місто для прогнозу погоди, оновлення прошивки, перезапуск модуля.
- Сторінка /*weather* – спеціальний маршрут, що повертає *JSON*-об’єкт з поточними погодними даними. Використовується *JavaScript* для періодичного оновлення даних.

Усі *HTML, CSS* та *JavaScript*-файли зберігаються у флеш-пам’яті мікроконтролера, завдяки файловій системі *LittleFS*. Це дозволяє модуль самостійно обслуговувати запити клієнта, не потребуючи зовнішнього сервера.

Реалізація веб-сервера

У прошивці *ESP*-01 реалізовано асинхронний веб-сервер з використанням бібліотеки *ESPAsyncWebServer*. Асинхронність дозволяє пристрою обслуговувати запити до веб-інтерфейсу без блокування основного циклу, що критично для пристроїв із низькою обчислювальною потужністю.

Основні кроки реалізації:

1. Ініціалізація *Wi-Fi* у режимі *STA* (станції).
2. Ініціалізація файлової системи *LittleFS*.
3. Запуск веб-сервера з маршрутизацією:
   - `server.on("/", ...)` – обслуговування головної сторінки.
   - `server.on("/weather", ...)` – видача поточних погодних даних у форматі *JSON*.
   - `server.on("/settings", HTTP_POST, ...)` – обробка змін налаштувань, надісланих із форми.Представлено на Рисунку 2.8

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(LittleFS, "/index.html", "text/html");
});

server.on("/weather", HTTP_GET, [](AsyncWebServerRequest *request){
  String json = getWeatherJson();  // функція формує JSON-рядок
  request->send(200, "application/json", json);
});
```

Рисунок 2.8- Асинхронний веб-сервер з використанням бібліотеки *ESPAsyncWebServer*

Дизайн інтерфейсу

Для користувача важливо, щоб інтерфейс був не лише функціональним, але й зручним у використанні. Було реалізовано:

- Мінімалістичний дизайн з використанням *CSS Flexbox* для гнучкого розташування блоків.
- Світла та темна тема (опційно).

- Іконки погоди – *SVG*-графіка, яка змінюється динамічно залежно від прогнозу (сонце, хмари, дощ тощо).
- Відображення статусу – індикатори підключення до *Wi-Fi*, помилок *API*, часу останнього оновлення.

Використано кастомні стилі, але структура дозволяє легко інтегрувати фреймворки типу *Bootstrap* за потреби.

Динамічні елементи та оновлення даних

Щоб уникнути постійного перезавантаження сторінки, для оновлення даних використано *JavaScript* та *AJAX*. Через кожні 60 секунд *JavaScript* скрипт виконує запит на `/weather`, отримує *JSON* і оновлює відповідні блоки *DOM* (температура, вологість, іконка тощо).Представлено на рисунку 2.9

```
fetch('/weather')
  .then(response => response.json())
  .then(data => {
    document.getElementById('temp').textContent = data.temp + "°C";
    document.getElementById('icon').src = "/icons/" + data.icon + ".svg";
  });
```

Рисунок 2.9- оновлення даних використовуючи *JavaScript* та *AJAX*

Завантаження інтерфейсу у мікроконтролер

Файли веб-інтерфейсу (*HTML*, *CSS*, *JS*, іконки) завантажуються у пам’ять *ESP*-01 за допомогою інструменту *ESP*8266 *LittleFS Uploade**r***, який інтегрується в *Arduino IDE*. Для цього:

1. Створено директорію `data/` поруч із файлом прошивки.
2. Всі веб-файли розміщено у цій директорії.
3. Обрано відповідну плату (*Generic ESP*8266 *Module*).
4. Через меню `Tools → ESP8266 LittleFS Data Upload` завантажено веб-файли до мікроконтролера.

Цей підхід дозволяє розділити логіку прошивки та веб-інтерфейсу, що спрощує оновлення окремих компонентів.

Тестування інтерфейсу

Після завантаження та запуску сервера проведено тестування в реальному середовищі. До пристрою підключено смартфон та комп’ютер у тій

же *Wi-Fi* мережі. В браузері відкрито *IP*-адресу пристрою (наприклад, `http://192.168.1.54`), що дозволило перевірити:

- швидкість завантаження сторінки;
- коректність відображення даних;
- обробку некоректних запитів;
- стабільність роботи сервера при кількох одночасних з’єднаннях.

У результаті реалізовано повноцінний, автономний веб-інтерфейс для *ESP*-01, що забезпечує інформування користувача про стан пристрою, дозволяє переглядати погоду та виконувати базові налаштування. Досягнуто високу продуктивність при мінімальних апаратних ресурсах.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Огляд середовища розробки *Arduino IDE*

Arduino IDE (*Integrated Development Environment*) є одним з найпопулярніших та найзручніших середовищ розробки для програмування мікроконтролерів сімейства *Arduino* та *ESP*. Це кросплатформенне програмне забезпечення, яке дозволяє розробникам створювати, компілювати та завантажувати програми на різноманітні мікроконтролери, включаючи *ESP*-01, який використовується в даному проекті розумного годинника.

Arduino IDE відрізняється простотою використання та інтуїтивно зрозумілим інтерфейсом, що робить його ідеальним вибором як для початківців, так і для досвідчених розробників. Середовище підтримує мову програмування *C*/*C++* з додатковими бібліотеками та функціями, спеціально адаптованими для роботи з мікроконтролерами.

Arduino IDE було вперше представлено в 2005 році як частина проекту *Arduino*, який мав на меті створення доступної платформи для прототипування електронних пристроїв. Початкова версія базувалася на *Processing IDE* та була написана на *Java*, що забезпечувало кросплатформенність. Протягом років *Arduino IDE* зазнало значних змін та покращень. Версії 1.0-1.6 включали базову функціональність та підтримку основних плат *Arduino*. Версії 1.6-1.8 принесли розширену підтримку сторонніх плат та покращений менеджер бібліотек. Версія 2.0 та новіші представляють повністю перероблений інтерфейс з автодоповненням коду та покращеною продуктивністю, базуючись на технологіях *Eclipse Theia* та *Language Server Protocol*.

Архітектура, компоненти та інтерфейс користувача

Arduino IDE має модульну архітектуру, яка складається з декількох ключових компонентів, що забезпечують повний цикл розробки програмного забезпечення для мікроконтролерів. Основне ядро (*Core*) відповідає за

основну функціональність середовища, включаючи управління проектами, компіляцію коду та взаємодію з апаратним забезпеченням. Воно написане на *Java* та використовує різноманітні бібліотеки для забезпечення кросплатформенності.

Текстовий редактор є центральним елементом *Arduino IDE* і підтримує підсвічування синтаксису для мови *Arduino* (*C*/*C*++), автоматичне відступлення коду, пошук та заміну тексту. В новіших версіях додано функції автодоповнення та інтелектуальні підказки, що значно полегшує процес написання коду.

Менеджер плат (*Board Manager*) дозволяє встановлювати та керувати підтримкою різних типів мікроконтролерів. Для роботи з *ESP*-01 необхідно встановити пакет *ESP*8266, який додає підтримку всієї лінійки мікроконтролерів *ESP*8266. Менеджер бібліотек (*Library Manager*) забезпечує легкий доступ до тисяч готових бібліотек, які розширюють функціональність мікроконтролерів. Бібліотеки можна встановлювати безпосередньо з інтерфейсу *IDE* або імпортувати вручну.

Компілятор та інструменти збірки використовують *GCC* компілятор для перетворення коду *C*/*C*++ в машинний код. Процес компіляції включає препроцесинг, компіляцію, асемблювання та лінкування, при цьому *Arduino IDE* автоматично керує всіма цими етапами.

Інтерфейс *Arduino IDE* спроектований з урахуванням принципів простоти та зручності використання. Головне меню містить всі основні команди для роботи з файлами, редагування коду, компіляції та завантаження програм. Меню "*Tools*" (Інструменти) особливо важливе, оскільки дозволяє вибирати тип плати, порт та інші налаштування. Панель інструментів розташована під головним меню та містить кнопки швидкого доступу до найбільш використовуваних функцій: перевірка коду, завантаження на плату, створення нового файлу, відкриття та збереження.

Область редактора займає центральну частину вікна, де відбувається написання та редагування коду. Підтримується табуляція для роботи з

декількома файлами одночасно. Консоль виведення розташована в нижній частині вікна та відображає результати компіляції, помилки, попередження та іншу службову інформацію. Монітор серійного порту є окремим вікном, яке дозволяє відправляти та отримувати дані через серійний порт, що надзвичайно корисно для налагодження програм.

Налаштування та особливості програмування *ESP*-01

Для успішної роботи з мікроконтролером *ESP*-01 необхідно виконати ряд специфічних налаштувань в *Arduino IDE*. Через менеджер плат необхідно встановити пакет "*ESP*8266 *by ESP*8266 *Community*", який додає підтримку всієї лінійки мікроконтролерів *ESP*8266, включаючи *ESP*-01. В меню "*Tools*" → "*Board*" потрібно вибрати "*Generic ESP*8266 *Module*" або "*ESP*-01" та налаштувати параметри: *CPU Frequency* на 80 *MHz, Flash Size* в залежності від версії *ESP*-01 (зазвичай 1*MB*), *Flash Mode* на *DIO* або *QIO*. Також необхідно вибрати правильний *COM*-порт, до якого підключений *ESP*-01 через *USB-UART* адаптер.

Програмування *ESP*-01 в *Arduino IDE* має особливості, пов'язані з архітектурою мікроконтролера *ESP*8266. Як і в класичних *Arduino* скетчах, програма для *ESP*-01 повинна містити функції `setup()` та `loop()`. Функція `setup()` виконується один раз при запуску і використовується для ініціалізації змінних, налаштування портів введення/виведення та запуску бібліотек. Функція `loop()` виконується циклічно після завершення `setup()` та містить основну логіку програми.

ESP-01 має вбудований *Wi-Fi* модуль, для роботи з яким використовується бібліотека *ESP*8266*WiFi*. Ця бібліотека надає *API* для підключення до *Wi-Fi* мереж, створення точок доступу та роботи з мережевими протоколами. Типовий код підключення до *Wi-Fi* включає виклик функцій `WiFi.begin()` з параметрами *SSID* та паролю мережі, після чого програма очікує встановлення з'єднання.

*ESP*8266 має обмежену кількість *RAM* (близько 80*KB* доступно для користувача), тому необхідно ретельно керувати використанням пам'яті.

Arduino IDE надає інструменти для моніторингу використання пам'яті під час компіляції, показуючи відсоток використання *Flash* та *RAM* пам'яті. *ESP*-01 також підтримує файлову систему *SPIFFS* (*SPI Flash File System*), яка дозволяє зберігати файли у флеш-пам'яті мікроконтролера, що корисно для зберігання конфігураційних файлів, веб-сторінок або інших даних.

Інструменти налагодження та бібліотеки

Arduino IDE надає декілька важливих інструментів для налагодження програм, які є критично важливими при розробці складних проектів, таких як розумний годинник. Серійний монітор є основним інструментом для налагодження, який дозволяє відправляти та отримувати дані через серійний порт. Він корисний для виведення діагностичної інформації, тестування функцій та моніторингу стану програми в реальному часі. Плоттер серійного порту є графічним інструментом для відображення числових даних у вигляді графіків, що особливо корисно для візуалізації показників датчиків або моніторингу змін параметрів у часі.

Виведення налагоджувальної інформації здійснюється за допомогою функцій `Serial.print()` та `Serial.println()`, які дозволяють виводити інформацію про стан програми, значення змінних, результати виконання функцій та повідомлення про помилки. Ці функції можна використовувати на різних етапах виконання програми для відстеження її роботи.

Arduino IDE надає доступ до величезної кількості бібліотек, багато з яких сумісні з *ESP*-01 та необхідні для реалізації функціональності розумного годинника. Стандартні бібліотеки *ESP*8266 включають *ESP*8266*WiFi* для роботи з *Wi-Fi*, *ESP*8266*WebServer* для створення веб-серверів, *ESP*8266*HTTPClient* для виконання *HTTP* запитів, *ArduinoJson* для роботи з *JSON* даними та *NTPClient* для синхронізації часу через інтернет.

Для роботи з дисплеями доступні спеціалізовані бібліотеки: *Adafruit SSD*1306 для *OLED* дисплеїв, *LiquidCrystal* для *LCD* дисплеїв та *U*8*g*2 як універсальна бібліотека для різних типів дисплеїв. Для роботи з зовнішніми

API корисними є бібліотеки *OpenWeatherMap* для отримання даних про погоду та *WiFiManager* для спрощення процесу підключення до *Wi-Fi* мереж.

Переваги, недоліки та альтернативні рішення

Arduino IDE має ряд значних переваг, які робили його популярним вибором для розробки проектів на мікроконтролерах. Головною перевагою є простота використання та інтуїтивний інтерфейс, що дозволяє швидко почати роботу навіть початківцям. Велика спільнота користувачів та розробників забезпечує постійну підтримку, розробку нових бібліотек та вирішення проблем. Широка підтримка різних мікроконтролерів, включаючи всю лінійку *ESP*, робить *Arduino IDE* універсальним інструментом. Багата бібліотека готових рішень дозволяє швидко реалізувати складну функціональність без написання коду з нуля. Кросплатформенність забезпечує роботу на *Windows*, *macOS* та *Linux*, а безкоштовність використання робить його доступним для всіх.

Водночас *Arduino IDE* має і недоліки. Обмежені можливості налагодження порівняно з професійними *IDE* можуть ускладнити розробку складних проектів. Відсутність розширених функцій рефакторингу коду та автоматичного аналізу може сповільнити розробку. Іноді спостерігається повільна робота з великими проектами, особливо при використанні багатьох бібліотек. Обмежені можливості керування версіями ускладнюють роботу в команді або підтримку декількох версій проекту.

Існують альтернативні середовища розробки, які можуть бути корисними в певних ситуаціях. *PlatformIO* є сучасним *IDE* на базі *Visual Studio Code* з розширеними можливостями налагодження, керування проектами та підтримкою різних платформ. *ESP-IDF* представляє офіційний фреймворк від *Espressif* для розробки на *ESP*8266/*ESP*32 з повним контролем над апаратним забезпеченням та доступом до низькорівневих функцій. *MicroPython* дозволяє програмувати *ESP*-01 на мові *Python*, що може бути зручніше для розробників, знайомих з цією мовою.

Arduino IDE залишається оптимальним вибором для розробки проекту розумного годинника на базі *ESP*-01 завдяки своїй простоті, широкій підтримці спільноти та наявності всіх необхідних інструментів та бібліотек. Середовище забезпечує повний цикл розробки - від написання коду до його компіляції та завантаження на мікроконтролер. Особливо важливими для даного проекту є можливості роботи з *Wi-Fi*, виконання *HTTP* запитів для отримання даних про погоду, управління дисплеєм та синхронізація часу.

Незважаючи на деякі обмеження у порівнянні з більш професійними середовищами розробки, *Arduino IDE* повністю задовольняє потреби даного проекту та дозволяє зосередитися на реалізації основної функціональності розумного годинника, а не на складнощах налаштування середовища розробки. Простота використання, велика кількість доступних бібліотек та активна спільнота роблять *Arduino IDE* ідеальним вибором для проектів *IoT* та розумних пристроїв на базі *ESP*-01.

3.2 Отримання поточної дати й часу через *NTP*

Загальні принципи роботи протоколу *NTP* та його застосування в *IoT* пристроях

Network Time Protocol (*NTP*) є одним з найстаріших та найнадійніших протоколів інтернету, призначеним для синхронізації часу між комп'ютерними системами через мережі з змінною затримкою. Розроблений в 1985 році Девідом Міллсом, *NTP* забезпечує точність синхронізації часу до мілісекунд у локальних мережах та до десятків мілісекунд через інтернет. Для *IoT* пристроїв, таких як розумний годинник на базі *ESP*-01, *NTP* є критично важливим компонентом, оскільки мікроконтролери зазвичай не мають власних годинників реального часу (*RTC*) або їх точність недостатня для практичного використання.

Протокол *NTP* працює за принципом клієнт-серверної архітектури, де клієнт (в нашому випадку *ESP*-01) надсилає запити на *NTP* сервери та отримує

точний час. *NTP* використовує ієрархічну систему серверів, організовану у страти (*Stratum*). Страт 0 представляють еталонні джерела часу, такі як атомні годинники або *GPS* приймачі. Страт 1 - це сервери, безпосередньо підключені до еталонних джерел. Страт 2 синхронізуються з серверами страт 1, і так далі. Ця ієрархічна структура забезпечує високу надійність та розподіляє навантаження між серверами.

Особливо важливим для *IoT* пристроїв є те, що *NTP* враховує мережеву затримку та може компенсувати її для більш точної синхронізації. Протокол вимірює час відправлення запиту, час отримання на сервері, час відправлення відповіді та час отримання відповіді клієнтом. На основі цих чотирьох часових міток *NTP* обчислює мережеву затримку та коригує час відповідно.

Реалізація *NTP* клієнта на *ESP*-01 з використанням спеціалізованих бібліотек

Для реалізації *NTP* клієнта на *ESP*-01 в *Arduino IDE* найчастіше використовуються дві основні бібліотеки: *NTPClient* та вбудовані функції *ESP*8266, такі як `configTime()`. Бібліотека *NTPClient* надає простий та зрозумілий *API* для роботи з *NTP* серверами, дозволяючи легко отримувати поточний час та форматувати його відповідно до потреб проекту.

Встановлення бібліотеки *NTPClient* здійснюється через менеджер бібліотек *Arduino IDE*. Після встановлення необхідно підключити відповідні заголовкові файли та створити об'єкт *NTPClient* з вказанням *NTP* сервера та часового поясу. Типова ініціалізація виглядає наступним чином. Приклад коду на рисунку 3.1

```cpp
#include <NTPClient.h>
#include <WiFiUdp.h>

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 7200
```

Рисунок 3.1– Реалізація *NTP* клієнта на *ESP*-01 з використанням спеціалізованих бібліотек

У цьому коді створюється *UDP* з'єднання для *NTP* запитів, ініціалізується клієнт з вказанням публічного *NTP* сервера *pool.ntp.org*, встановлюється зміщення часового поясу в секундах (7200 секунд = 2 години для України в зимовий час) та інтервал оновлення в мілісекундах (60000 мс = 1 хвилина).

Альтернативним підходом є використання вбудованих функцій *ESP*8266. Функція `configTime()` дозволяє налаштувати автоматичну синхронізацію часу з *NTP* серверами без використання додаткових бібліотек. Цей підхід інтегрується з системними функціями мікроконтролера та може бути більш ефективним з точки зору використання пам'яті. Приклад коду на рисунку 3.2

```cpp
#include <time.h>

configTime(7200, 0, "pool.ntp.org", "time.nist.go
```

Рисунок 3.2 – Функція `configTime()`

Ця функція автоматично налаштовує системний час *ESP*8266, після чого можна використовувати стандартні функції *C* для роботи з часом, такі як `time()`, `localtime()` та `strftime()`.

Вибір оптимальних *NTP* серверів та налаштування часових поясів

Вибір правильних *NTP* серверів є критично важливим для забезпечення надійної та точної синхронізації часу. Існує кілька категорій *NTP* серверів, кожна з яких має свої переваги та недоліки. Публічні пули *NTP* серверів, такі як *pool.ntp.org*, є найпопулярнішим вибором для більшості проектів. Вони забезпечують високу доступність завдяки географічному розподілу серверів та автоматичному балансуванню навантаження.

Для проектів, що розгортаються в Україні, рекомендується використовувати географічно близькі сервери для мінімізації мережевої затримки. Наприклад, "*ua.pool.ntp.org*" для українських серверів або

"*europe.pool.ntp.org*" для європейських. Також можна використовувати конкретні сервери, такі як "*time.nist.gov*" (США) або "*ptbtime*1.*ptb.de*" (Німеччина), але це може знизити надійність через залежність від конкретного сервера.

Важливим аспектом є правильне налаштування часового поясу. Україна знаходиться в часовому поясі *UTC*+2 взимку та *UTC*+3 влітку (з урахуванням переходу на літній час). При використанні бібліотеки *NTPClient* зміщення часового поясу вказується в секундах: 7200 секунд для зимового часу (*UTC*+2) та 10800 секунд для літнього часу (*UTC*+3).

Для автоматичного врахування переходу на літній час можна реалізувати логіку, яка визначає поточний період року та встановлює відповідне зміщення. Це можна зробити на основі дати або використовуючи правила переходу на літній час, які діють в Україні (зазвичай останні неділі березня та жовтня).

Рекомендується також налаштувати резервні *NTP* сервери для підвищення надійності. У випадку недоступності основного сервера система може автоматично переключитися на резервний. Це особливо важливо для пристроїв, які повинні працювати безперебійно. Приклад коду на рисунку 3.3

```cpp
NTPClient timeClient(ntpUDP, "ua.pool.ntp.org", 7200, 60000);
// Резервні сервери можна реалізувати через масив серверів
const char* ntpServers[] = {"ua.pool.ntp.org", "pool.ntp.org", "time.nist.gov"};
```

Рисунок 3.3 – Резервні *NTP* сервери для підвищення надійності.

Обробка помилок синхронізації та забезпечення надійності отримання часу

Робота з мережевими протоколами завжди пов'язана з можливістю виникнення помилок, тому важливо реалізувати надійну систему обробки помилок та відновлення після збоїв. Основні типи помилок при роботі з *NTP*

включають: відсутність мережевого з'єднання, недоступність *NTP* сервера, мережеві затримки та часові розбіжності.

Для виявлення та обробки помилок синхронізації необхідно реалізувати систему перевірок та повторних спроб. При використанні бібліотеки *NTPClient* можна перевіряти успішність оновлення часу та реагувати на помилки відповідним чином. Приклад коду на рисунку 3.4

```cpp
void updateTimeWithRetry() {
    int retryCount = 0;
    const int maxRetries = 3;

    while (retryCount < maxRetries) {
        if (timeClient.update()) {
            Serial.println("Time updated successfully");
            lastSyncTime = millis();
            return;
        } else {
            retryCount++;
            Serial.print("Time sync failed, retry ");
            Serial.println(retryCount);
            delay(5000); // Затримка перед повторною спробою
        }
    }

    Serial.println("Failed to sync time after all retries");
    // Встановити прапорець помилки синхронізації
    timeSyncError = true;
}
```

Рисунок 3.4 – Обробка помилок синхронізації та забезпечення надійності отримання часу

Важливо також відстежувати час останньої успішної синхронізації та періодично оновлювати час. *ESP*-01 має внутрішній годинник, який може підтримувати відносно точний час протягом кількох годин після синхронізації, але його точність знижується з часом через дрейф кварцового резонатора.

Рекомендується реалізувати адаптивну стратегію синхронізації, яка враховує якість мережевого з'єднання та точність попередніх синхронізацій. При стабільному з'єднанні можна збільшити інтервал між синхронізаціями, а при нестабільному - зменшити його.

Для критично важливих застосувань можна реалізувати локальне зберігання часу у *EEPROM* або *SPIFFS*, щоб зберегти приблизний час навіть після перезавантаження пристрою. Приклад коду на рисунку 3.5.

```cpp
void saveTimeToEEPROM() {
    unsigned long currentTime = timeClient.getEpochTime();
    EEPROM.put(0, currentTime);
    EEPROM.commit();
}

unsigned long loadTimeFromEEPROM() {
    unsigned long storedTime;
    EEPROM.get(0, storedTime);
    return storedTime;
}
```

Рисунок 3.5 – Локальне зберігання часу у *EEPROM*

Форматування та відображення дати й часу для розумного годинника

Отримавши точний час через *NTP*, необхідно правильно відформатувати його для відображення на дисплеї розумного годинника. Бібліотека *NTPClient* надає кілька зручних методів для отримання різних компонентів часу: `getHours()`, `getMinutes()`, `getSeconds()`, `getDay()`, `getMonth()` та інші.

Для створення зручного для користувача інтерфейсу важливо правильно форматувати час з урахуванням локальних традицій. В Україні прийнято використовувати 24-годинний формат часу та формат дати ДД.ММ.РРРР. Типова функція форматування може виглядати наступним чином. Приклад коду на рисунку 3.6

```cpp
String getFormattedTime() {
    int hours = timeClient.getHours();
    int minutes = timeClient.getMinutes();
    int seconds = timeClient.getSeconds();

    String timeString = "";
    if (hours < 10) timeString += "0";
    timeString += String(hours) + ":";
    if (minutes < 10) timeString += "0";
    timeString += String(minutes) + ":";
    if (seconds < 10) timeString += "0";
    timeString += String(seconds);

    return timeString;
}

String getFormattedDate() {
    time_t epochTime = timeClient.getEpochTime();
    struct tm *ptm = gmtime(&epochTime);

    int currentDay = ptm->tm_mday;
    int currentMonth = ptm->tm_mon + 1;
    int currentYear = ptm->tm_year + 1900;

    String dateString = "";
    if (currentDay < 10) dateString += "0";
    dateString += String(currentDay) + ".";
    if (currentMonth < 10) dateString += "0";
    dateString += String(currentMonth) + ".";
    dateString += String(currentYear);

    return dateString;
}
```

Рисунок 3.6 – Форматування та відображення дати й часу

Оптимізація енергоспоживання та управління частотою синхронізації

Оскільки *ESP*-01 може використовуватися в проектах з обмеженим живленням, важливо оптимізувати енергоспоживання при роботі з *NTP*. Синхронізація часу вимагає активного *Wi-Fi* з'єднання, яке споживає значну кількість енергії. Тому необхідно знайти баланс між точністю часу та енергоефективністю.

Рекомендується використовувати адаптивну частоту синхронізації, яка залежить від точності внутрішнього годинника *ESP*-01 та стабільності мережевого з'єднання. При стабільній роботі можна збільшити інтервал між синхронізаціями до 1-2 годин, при проблемах з мережею - зменшити до 15-30 хвилин.

Для додаткової економії енергії можна реалізувати режим глибокого сну (*deep sleep*) між синхронізаціями, особливо якщо годинник не потребує постійного відображення інформації. Приклад коду на рисунку 3.7

```cpp
void enterDeepSleep(unsigned long sleepTime) {
    Serial.println("Entering deep sleep...");
    ESP.deepSleep(sleepTime * 1000000); // Переведення секунд в мікросекунди
}
```

Рисунок 3.7 – Режим глибокого сну (*deep sleep*)

Реалізація синхронізації часу через *NTP* протокол є ключовим компонентом розумного годинника на базі *ESP*-01. Правильна реалізація *NTP* клієнта забезпечує точний та надійний час, необхідний для коректної роботи всіх функцій пристрою. Використання спеціалізованих бібліотек, таких як *NTPClient*, значно спрощує розробку та забезпечує стабільну роботу.

Важливими аспектами успішної реалізації є правильний вибір *NTP* серверів, налаштування часових поясів, обробка помилок синхронізації та оптимізація енергоспоживання. Адаптивна стратегія синхронізації дозволяє забезпечити баланс між точністю часу та ефективністю використання ресурсів.

Реалізована система синхронізації часу створює надійну основу для всіх інших функцій розумного годинника, включаючи відображення поточного часу, планування подій та отримання даних про погоду з прив'язкою до конкретного часу.

3.3 Отримання прогнозу погоди через *API*

Одним із ключових функціональних елементів проєкту «розумного годинника» є можливість відображення актуального прогнозу погоди в режимі реального часу. У світі Інтернету речей (*IoT*) така функція набуває особливої актуальності, адже забезпечує користувача не лише зручним доступом до метеоінформації, а й дозволяє адаптувати власну активність відповідно до погодних умов. Для реалізації цього функціоналу було інтегровано доступ до онлайн-сервісу *WeatherAPI.com*, який надає широкий спектр погодних даних через стандартний *RESTful API*.

Загальна характеристика *WeatherAPI*

WeatherAPI є одним із провідних онлайн-сервісів для надання погодної інформації, який характеризується високою точністю, широким набором параметрів і підтримкою зручного формату передачі даних — *JSON*. Сервіс підтримує як безкоштовні, так і платні тарифні плани. Навіть на безкоштовному рівні користувач отримує змогу надсилати значну кількість запитів щодня, чого цілком достатньо для невеликих *IoT*-проєктів.

Після реєстрації користувачу надається унікальний ключ *API*, за допомогою якого можна ідентифікувати всі запити. У відповіді *API* повертається детальний прогноз погоди, а також поточні метеоумови у зазначеній локації. Завдяки цьому сервісу було реалізовано оновлення метеоданих на екрані годинника з певною періодичністю.

Принцип інтеграції з мікроконтролером *ESP*-01

Для отримання прогнозу погоди мікроконтролер *ESP*-01 підключається до *Wi-Fi*-мережі, після чого ініціює *HTTP*-запит до сервера *WeatherAPI*. У тілі запиту зазначається ключ *API*, а також параметри місцезнаходження — місто або координати. Відповідь приходить у форматі *JSON*, що зручно для обробки на мікроконтролері за допомогою бібліотек на кшталт `ArduinoJson`.

Найважливішим аспектом є правильна побудова *URL*-адреси запиту та обробка відповіді. Система повинна не лише коректно зчитувати значення, але

й фільтрувати зайві поля, адаптуючи дані до обмежених ресурсів *ESP*-01 (обсяг оперативної пам’яті та обчислювальна потужність мікроконтролера невеликі).

Набір отримуваних метеоданих

У межах дипломного проєкту було реалізовано обробку та вивід наступних параметрів, які отримуються через *WeatherAPI*:

- Температура повітря — значення повертається у градусах Цельсія (наприклад, `temp_c: 22.4`). Це один із базових індикаторів погодних умов, який відображається у центральній частині екрана пристрою.
- Дата та локальний час — важливою функцією *API* є підтримка часових поясів, завдяки чому повертається точний час для заданого міста (поле `location.localtime`). Це дозволяє не використовувати окремі модулі реального часу (*RTC*), а отримувати актуальний час безпосередньо з *API*.
- Стан погоди (ясно, похмуро, дощ тощо) — *API* повертає як текстовий опис, так і код/іконку погодних умов. Наприклад: `condition.text: "Partly cloudy"`. Це значення використовується для виводу символів або кастомних іконок на *OLED*-дисплеї.
- Вологість повітря — виражається у відсотках і дозволяє визначити рівень насичення повітря водяною парою. Висока або низька вологість впливає на комфорт користувача.
- Швидкість та напрямок вітру — значення повертаються у кілометрах на годину (`wind_kph`) і градусах або текстових позначеннях (наприклад, `NW`), що дає змогу орієнтувати користувача щодо погодних трендів.
- Атмосферний тиск — значення `pressure_mb` повертається у мілібарах. Цей параметр може слугувати ознакою зміни погоди та впливати на самопочуття метеозалежних людей.
- Якість повітря (*Air Quality Index*) — через окремий модуль *API* повертається індекс якості повітря (*AQI*), а також значення таких

параметрів, як *PM*2.5, *PM*10, рівень *CO*, NO_2, SO_2 тощо. У пристрої реалізовано вивід текстової оцінки якості повітря — «*Good*», «*Moderate*», «*Unhealthy*» тощо.

Переваги використання *WeatherAPI*

На відміну від інших сервісів, таких як *OpenWeatherMap* або *AccuWeather*, саме *WeatherAPI* забезпечує ширший спектр даних у межах безкоштовного тарифу, а також простішу структуру *JSON*-відповіді, що є критично важливим для мікроконтролерів із обмеженими ресурсами. Крім того, сервіс підтримує щогодинне оновлення даних, що дозволяє налаштувати пристрій на періодичні звернення без перевищення денного ліміту запитів.

Інтеграція *WeatherAPI* значно підвищила практичну цінність розумного годинника, оскільки дозволила зробити пристрій не лише естетичним і технологічним, а й максимально інформативним. Користувач отримує доступ до метеоумов у будь-який момент часу, без необхідності використовувати смартфон або зовнішні ресурси.

Підключення та використання *API* сервісу погоди — це приклад ефективної взаємодії *IoT*-пристрою з хмарною інфраструктурою. *WeatherAPI* забезпечує достатній рівень надійності, точності й зручності інтеграції, а розумний годинник, у свою чергу, демонструє здатність працювати з великим обсягом зовнішньої інформації в умовах обмежених ресурсів. Отримані в результаті дані не лише збагачують функціонал пристрою, але й роблять його корисним у повсякденному використанні.

3.4 Виведення інформації на дисплей

Однією з найважливіших задач у реалізації функціонального розумного годинника є виведення зібраної або обробленої інформації на екран. Саме дисплей є головним каналом взаємодії між пристроєм та користувачем. Завдяки йому можна передати не лише час або дату, але й прогноз погоди, повідомлення, статус *Wi-Fi*, індикатори якості повітря та інші важливі

параметри. Для реалізації цієї функції в дипломному проєкті було використано *OLED*-дисплей на базі контролера *SSD*1306 з роздільною здатністю 128*x*64 пікселі.

Вибір дисплея

Вибір саме *OLED*-дисплея (*Organic Light Emitting Diode*) є виправданим з кількох причин. По-перше, ці дисплеї мають високу контрастність: пікселі, які не активні, не підсвічуються, що дає глибокий чорний фон. По-друге, низьке енергоспоживання робить *OLED*-екрани ідеальними для автономних або малопотужних *IoT*-пристроїв. По-третє, невеликі габарити та можливість адресного керування кожним пікселем дозволяють створювати як текстову, так і графічну інформацію.

У рамках проєкту було використано дисплей із діагоналлю 0.96 дюйма, який має інтерфейс *I²C*. Такий інтерфейс потребує лише двох ліній зв'язку — *SDA* (дані) та *SCL* (тактовий сигнал), що є важливою перевагою при роботі з мікроконтролером *ESP*-01, який має обмежену кількість доступних *GPIO*-пінів.

Підключення та ініціалізація

Оскільки *ESP*-01 має лише два доступні *GPIO*-піни (*GPIO*0 та *GPIO*2), підключення дисплея по *I²C* є найбільш доцільним варіантом. Для забезпечення живлення використовується напруга 3.3 В, сумісна з дисплеєм та *ESP*-01. Сигнальні лінії зазвичай підключаються до *GPIO*0 (*SDA*) та *GPIO*2 (*SCL*), хоча для нестандартних розкладок можуть бути використані програмні бібліотеки з підтримкою "софтверного *I*2*C*".

У програмному коді ініціалізація дисплея відбувається за допомогою бібліотеки *Adafruit_SSD*1306, яка забезпечує зручні методи для виводу тексту, зображень, ліній, графіків та кастомних іконок.

Структура виводу інформації

Оскільки розмір дисплея обмежений, важливо продумати логічну структуру відображення даних, щоб користувач міг з першого погляду

зрозуміти, що саме показує екран. У дипломному проєкті було реалізовано циклічний або умовний вивід наступних блоків інформації:

- Дата та поточний час — виводяться у верхній частині дисплея. Формат: `04.06.2025 14:23`.
- Температура повітря — одне з найбільш помітних значень, розміщується по центру або зліва великим шрифтом.
- Стан погоди — підпис: «Ясно», «Хмарно», «Дощ» тощо. Додатково супроводжується умовною іконкою (хмара, сонце тощо), яку відображено через побітову графіку.
- Вологість, тиск, вітер — ці параметри згруповано й виводяться у нижній частині дисплея. Для економії місця використовуються скорочення: *Hum*: 45%, *Pres*: 1012, *Wind*: 12 *km/h*.
- *Я*кість повітря (*AQI*) — у вигляді оцінки «*Good*», «*Moderate*» або індикатора кольором (можливо, через спеціальний символ).
- Циклічність відображення інформації передбачає автоматичну зміну екранів через певний інтервал часу (наприклад, кожні 5 секунд) або в залежності від натискання кнопки, якщо вона реалізована.

Відображення іконок та графіки

Одним із ключових аспектів є графічне оформлення погодних умов. Створення іконок для сонця, хмари, дощу, снігу тощо реалізується за допомогою масивів байтів, що відповідають побітовому малюнку. *OLED*-дисплей *SSD*1306 підтримує вивід графіки через функції *drawBitmap*(), які приймають координати, масив байтів та розміри іконки.

Також було реалізовано просту анімацію (наприклад, рух хмар або зміна іконки в залежності від *API*-даних), що надає пристрою динамічності.

Обмеження й оптимізація

Робота з дисплеєм у контексті мікроконтролера *ESP*-01 має низку обмежень:

- Малий обсяг оперативної пам’яті (менше 8 КБ для користувача): не дозволяє одночасно зберігати великі графічні масиви, тому було впроваджено оптимізовану систему — зберігання іконок у *PROGMEM* (постійній пам’яті).
- Відсутність багатозадачності: довелося балансувати між оновленням дисплея, збереженням з'єднання з *Wi-Fi* та опитуванням *API*.
- Час виводу: повна перерисовка дисплея може займати до 100 мс, що потребує обмеження частоти оновлення екрану.
- Для покращення ефективності використовувалися буферизовані методи роботи з дисплеєм — інформація формувалася у внутрішньому буфері, а потім передавалась на дисплей одразу.

Значення дисплея для *UX*

Екран — це обличчя пристрою. Саме через нього користувач отримує всю інформацію, не підключаючи додаткові пристрої. Завдяки *OLED*-дисплею з високим контрастом та грамотному дизайну інтерфейсу, розумний годинник виглядає не тільки функціонально, а й естетично привабливо. Використання кастомних іконок додає унікальності, а логічне групування інформації робить пристрій зручним навіть для користувачів без технічної підготовки.

3.5 Розробка алгоритмів роботи розумного годинника

Проєктування логіки роботи розумного годинника ґрунтується на створенні стабільної, ефективної та адаптивної програмної архітектури, яка дозволяє безперебійно функціонувати пристрою в умовах реального середовища. Оскільки пристрій має виконувати одночасно кілька задач — синхронізацію часу, підключення до мережі, отримання даних з *API*, їх обробку, вивід на дисплей та реагування на збої, — надзвичайно важливо продумати взаємодію всіх програмних компонентів, дотримуючись принципів асинхронної обробки подій.

Алгоритм основного циклу мікроконтролера побудований із використанням неблокуючої логіки, яка виключає застосування затримок, що можуть зупинити виконання інших функцій. Ключову роль у цьому відіграє функція `millis()`, яка дозволяє визначати момент, коли необхідно оновити ті чи інші дані. Такий підхід дає змогу оновлювати годинник, дані з погодного *API* та відображення на екрані у незалежному порядку, уникнувши перевантаження чи зависання мікроконтролера. Наприклад, синхронізація годинника може відбуватись кожні 30 хвилин, оновлення прогнозу погоди — щогодини, а зміна дисплейної інформації — кожні 10 секунд.

Основною умовою функціонування пристрою є наявність стабільного *Wi-Fi*-з'єднання. У випадку його втрати контролер продовжує працювати в автономному режимі, зберігаючи останні отримані дані. Алгоритм перевірки з'єднання виконується з певною періодичністю, і при виявленні стабільного сигналу відбувається повторна спроба авторизації в мережі. Завдяки використанню *EEPROM* або змінних у оперативній пам’яті можна уникнути повного скидання стану пристрою, навіть якщо з’єднання буде відсутнє протягом тривалого часу.

Синхронізація з сервером точного часу (*NTP*) дозволяє забезпечити точне відображення часу незалежно від тривалості роботи пристрою. Після підключення до Інтернету, *ESP*-01 надсилає *UDP*-запит до одного з публічних *NTP*-серверів, наприклад `pool.ntp.org`. В отриманій відповіді міститься час у форматі *UNIX Timestamp*, який конвертується в години, хвилини, секунди, а також дату. Важливо враховувати часовий пояс — ця поправка вноситься вручну в програмному коді. Якщо синхронізація тимчасово неможлива, використовується локальний таймер годинника до наступної успішної спроби з'єднання.

Після встановлення з'єднання з мережею наступним кроком є формування запиту до погодного *API*. У цьому проєкті використано сервіс *weatherapi.com*, що дозволяє отримувати детальні погодні дані у форматі *JSON*. У запиті вказується ключ доступу (*API Key*) та параметри, як-от назва

міста, координати або інші змінні. У відповідь повертається структурований *JSON*, з якого витягуються лише ті значення, що необхідні для відображення: температура повітря, вологість, тиск, швидкість і напрямок вітру, стан неба, якість повітря, дата та час останнього оновлення. Приклад інтерфейсу представлено на рисунку 3.8

Рисунок 3.8 – Відображення часу та дати

Важливо зазначити, що всі дані, отримані від *API*, повинні бути попередньо оброблені перед виводом. Температура, наприклад, округлюється до цілого числа, вологість подається у відсотках, а якість повітря оцінюється відповідно до шкали *AQI*, яку перетворюють на текстову інтерпретацію: "Добра", "Помірна", "Погана" тощо. Окремий блок коду відповідає за логіку перетворення напрямку вітру з градусів у словесний опис — наприклад, 0–45° відповідає напрямку "Північний", 46–90° — "Північно-Східний" і так далі. Це значно полегшує сприйняття даних користувачем. Приклад інтерфейсу представлено на рисунку 3.9.

Рисунок 3.9 – Відображення температури та вологості повітря

Інформація виводиться на *OLED*-дисплей за принципом циклічної ротації. Кожен екран відображає певний тип даних протягом 5–10 секунд: температура та стан неба, вологість і тиск, якість повітря та напрямок вітру, поточний час і дата. Така циклічна подача інформації не лише зручна для користувача, а й дозволяє уникнути статичності дисплея, що знижує ризик вигоряння пікселів. Усі текстові повідомлення супроводжуються відповідними іконками (наприклад, сонце, хмара, дощ), що дозволяє сприймати інформацію візуально на інтуїтивному рівні. Приклад інтерфейсу представлено на рисунках 3.10 – 3.11.

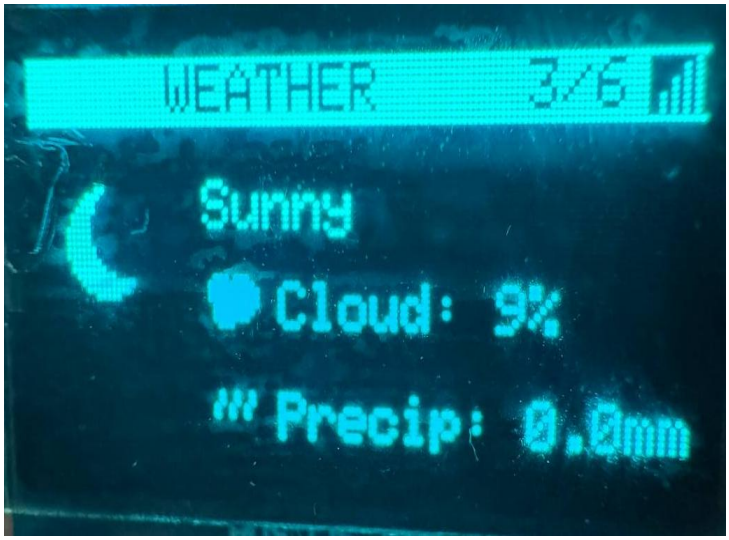

Рисунок – 3.10 Відображення поточної погоди, хмарності та опадів

Рисунок 3.11 – Відображення швидкості вітра його напряму та тиску

Для збереження стабільності пристрою реалізовано обробку виняткових ситуацій. Якщо з’єднання з *API* не вдалося, або *JSON* не вдалося розпарсити, на дисплей виводиться повідомлення про помилку (наприклад, "Помилка

з’єднання" або "Немає даних"). Якщо зафіксовано критичний збій у функціонуванні, пристрій автоматично перезапускається за допомогою механізму *Watchdog Timer* — вбудованої функції мікроконтролера, що відслідковує зависання коду. Цей підхід гарантує, що годинник не зависне у невизначеному стані.

Уся логіка проєкту реалізована з урахуванням можливості масштабування. Завдяки структурованості коду додавання нових функцій (наприклад, прогнозу на кілька днів, графіків змін температури, сенсорів фізичних показників) не потребує повної переробки архітектури. Достатньо інтегрувати нові модулі у вже існуючу систему оновлення й виводу даних.

Таким чином, алгоритми, реалізовані в рамках цього проєкту, дозволяють забезпечити повноцінну, стабільну та інформативну роботу розумного годинника. Система адаптується до змін середовища, працює надійно навіть за умов нестабільного Інтернету, а користувач отримує зручний і наочний інтерфейс для моніторингу погодних і часових показників.

3.6 Тестування та аналіз роботи пристрою в реальних умовах

Після завершення розробки апаратної частини та програмного забезпечення розумного годинника одним із ключових етапів стало тестування пристрою в реальних умовах експлуатації. Це дозволило оцінити не лише стабільність і правильність роботи системи, а й виявити слабкі місця, пов'язані з якістю з’єднання, точністю прогнозів, споживанням електроенергії та зручністю інтерфейсу.

Першим етапом тестування стала перевірка працездатності пристрою після ввімкнення. Годинник коректно ініціалізувався, виконував підключення до *Wi-Fi* мережі та отримував час через *NTP*. Випадки, коли сигнал *Wi-Fi* був слабкий або повністю відсутній, також були передбачені: пристрій демонстрував повідомлення про помилку або зберігав попередні дані, що підтверджувало працездатність механізмів резервного відображення.

Середній час підключення до мережі після запуску становив від 5 до 15 секунд залежно від якості сигналу. У випадках зміни мережі або її перезавантаження *ESP*-01 відновлював з'єднання самостійно, без ручного втручання.

Важливою частиною тестування стало оновлення даних з погодного *API*. Було перевірено стійкість роботи при відсутності з'єднання із сервером *WeatherAPI* або при надходженні некоректного *JSON*. У таких випадках пристрій переходив у режим очікування, повторюючи спробу через певний проміжок часу. За відсутності даних на дисплеї з'являлося повідомлення про помилку, яке не заважало роботі інших модулів пристрою. Це свідчило про високу стабільність програмної логіки та правильно реалізовану обробку виняткових ситуацій.

Наступним аспектом було тестування коректності виводу інформації на дисплей. Пристрій безпомилково відображав температуру, стан неба, вологість, якість повітря, швидкість і напрямок вітру, а також дату й час. Інформація подавалась циклічно — зміна екранів відбувалась із заданим інтервалом. Тестування тривало протягом кількох діб, зокрема вночі, щоб перевірити поведінку *OLED*-дисплея в умовах тривалого навантаження. Жодних збоїв у роботі дисплея не зафіксовано, хоча для зменшення ризику вигоряння пікселів у подальшому рекомендовано впровадити функцію автояскравості або автоматичного вимкнення екрана при бездіяльності.

Особлива увага під час тестування приділялася точності отриманих даних. Прогноз погоди та інші параметри, отримані з *WeatherAPI*, порівнювалися з показниками локальних метеосервісів і офіційних джерел. Відхилення температури не перевищувало 1–2°*C*, що є прийнятним для побутових пристроїв. Дані про вологість, тиск та якість повітря відповідали фактичним показникам з незначними варіаціями. Це підтвердило доцільність використання даного *API* для інтеграції в *IoT*-проєкти, де не вимагається лабораторна точність, але потрібна стабільність і оперативність.

Окремо було проведено спостереження за енергоспоживанням пристрою. З урахуванням використання *ESP*-01, *OLED*-дисплея та циклічного

обміну даними через *Wi-Fi* середній струм споживання становив близько 80–100 мА. При використанні пауербанку об’ємом 2000 мА·год пристрій міг працювати автономно близько 18–20 годин. Це вказує на необхідність оптимізації енергоспоживання у разі застосування пристрою в умовах відсутності постійного живлення.

У процесі тестування також оцінювалася зручність взаємодії користувача з пристроєм. Зчитування інформації з дисплея не викликало труднощів — шрифт було підібрано з урахуванням роздільної здатності екрану, а використання графічних іконок спрощувало сприйняття. Протягом тестування користувачі відзначили інтуїтивність інтерфейсу, хоча були запропоновані покращення, зокрема додати індикацію рівня сигналу *Wi-Fi* або змінити черговість виводу деяких даних.

У підсумку, проведене тестування підтвердило працездатність, стабільність та придатність розумного годинника до щоденного використання. Алгоритми оновлення, виводу та обробки даних працювали надійно, пристрій не потребував частих перезапусків або ручного втручання. Усі критичні ситуації (втрата мережі, збої *API*, перепади напруги) оброблялися без зупинки системи, що свідчить про грамотне технічне рішення на етапі розробки. Отже, створений пристрій успішно пройшов апробацію в реальних умовах і може бути використаний як основа для подальших *IoT*-розробок, орієнтованих на побутові або навчальні завдання.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці розумного годинника на базі мікроконтролера *ESP*-01 з можливістю відображення прогнозу погоди. У процесі виконання роботи було досліджено апаратні та програмні рішення, що дозволяють створити компактний пристрій для моніторингу метеоумов у режимі реального часу.

Було встановлено, що попит на пристрої з елементами Інтернету речей (*IoT*), які надають корисну інформацію користувачам у зручному вигляді, зростає. Розумний годинник, розроблений у рамках цієї роботи, реалізує функції відображення поточної дати, часу, температури, вологості, атмосферного тиску, якості повітря, швидкості та напрямку вітру, а також загального опису погодних умов (сонячно, похмуро, дощ тощо).

У ході дослідження було проаналізовано популярні рішення на основі *ESP*-01, а також проведено порівняльний аналіз доступних *API* для отримання метеоданих. Вибір було зупинено на сервісі *WeatherAPI*, який забезпечує зручну та надійну інтеграцію через *HTTP*-запити.

На основі обраного мікроконтролера *ESP*-01 і *OLED*-дисплея розроблено схему пристрою, що забезпечує наочне виведення інформації. Також було реалізовано алгоритми запиту та обробки даних з *API*, форматування результатів та оптимізацію відображення на екрані. Веб-інтерфейс дозволяє зручно змінювати налаштування пристрою за допомогою *Wi-Fi*-з'єднання.

Усі компоненти було протестовано в реальних умовах, що дозволило оцінити стабільність зв'язку, точність прогнозу, швидкість оновлення інформації та зручність користування. Пристрій успішно виконує поставлені завдання, демонструючи перспективність використання недорогих мікроконтролерів у побутових *IoT*-проєктах.

Розроблена система є компактною, енергоефективною та легко масштабованою, що дає змогу розглядати її як прототип для подальших

удосконалень, зокрема — розширення функціоналу, додавання сенсорів або розробка мобільного застосунку.

Таким чином, результати цієї кваліфікаційної роботи підтверджують доцільність використання платформ *IoT* для створення інформативних пристроїв з прогнозом погоди, що можуть бути використані в освітніх, побутових або хобі-проєктах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

Додано примітку [ООЗ]: Перелік має налічувати не менше 15 джерел

1. Зильберман С. В. *Arduino та електроніка: практичний посібник*. — К.: Фабула, 2020. — 248 с.
2. Дьяків М. С., Соколов С. Г. *Інтернет речей: архітектура, протоколи та безпека*. — К.: Діалектика, 2021. — 312 с.
3. *Klaus M. ESP8266 . — Amazon Digital Services LLC, 2019. — 130 p.*
4. *John Boxall. : - 65 . — No Starch Press, 2013. — 392 p.*
5. *WeatherAPI Official Documentation. — https://www.weatherapi.com/docs/*
6. *Espressif Systems. ESP8266EX . — https://www.espressif.com/sites/default/files/documentation/esp8266ex_datasheet_en.pdf*
7. *Adafruit. SSD1306 . — https://learn.adafruit.com/monochrome-oled-breakouts/arduino-library-and-examples*
8. *Bassi A. : . — Springer, 2022. — 274 p*
9. Programming ESP8266 with Arduino IDE [Електронний ресурс]. – Режим доступу: https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/ – Дата звернення: 09.05.2025.
10. Підключення OLED-дисплею до ESP8266. Гайд [Електронний ресурс]. – Режим доступу: https://alexgyver.ru/oled-display/ – Дата звернення: 11.05.2025.
11. ESP8266 Weather Station. Project Tutorial [Електронний ресурс]. – Режим доступу: https://www.instructables.com/ESP8266-Weather-Station/ – Дата звернення: 22.05.2025.
12. IoT Projects with ESP8266. Book by A. Sharma. – Packt Publishing, 2020. – 280 с.

13. Getting Started with the ESP-01 Wi-Fi Module [Електронний ресурс]. – Режим доступу: https://makeradvisor.com/getting-started-esp01/ – Дата звернення: 08.05.2025.

14. Інтернет речей на базі Arduino: навчальний посібник / І.М. Білан, В.О. Гордієнко. – К.: Ліра-К, 2021. – 192 с.

15. Вступ до роботи з OpenWeatherMap API. [Електронний ресурс]. – Режим доступу: https://dev.to/chrisdothtml/openweathermap-api-how-to-use-it-3g86 – Дата звернення: 22.05.2025.

ДОДАТОК А

Скетч програми головного модуля

```
#include <ESP8266WiFi.h>
```

```
#include <ESP8266WebServer.h>

#include <EEPROM.h>

#include <ESP8266mDNS.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>

#include <time.h>

#include <ESP8266HTTPClient.h>

#include <ArduinoJson.h>

#include <WiFiClient.h>


const char *WEATHER_API_KEY =
"218837b4535b4ea4857221451251505"; // Ключ API WeatherAPI


#include "types.h"

#include "oled_display.h"

#include "web_interface.h"


// --- Константи ---

#define AP_SSID "SmartClock_Setup" // SSID точки доступу

#define AP_PASSWORD ""             // Пароль точки доступу
(порожній = відкрита)


#define WIFI_CONNECT_TIMEOUT 60    // Таймаут підключення
Wi-Fi (с)

#define WIFI_RECONNECT_INTERVAL 30 // Інтервал
перепідключення Wi-Fi (с)
```

```
#define EEPROM_CONFIG_START_ADDR 0     // Адреса
конфігурації в EEPROM

#define EEPROM_SIZE sizeof(ConfigData) // Розмір
конфігурації



#define WEATHER_API_LOCATION "47.91,33.39" // Координати
для погоди

String weatherApiUrl =
"http://api.weatherapi.com/v1/current.json?key=" +
String(WEATHER_API_KEY) + "&q=" +
String(WEATHER_API_LOCATION) + "&aqi=yes";

#define WEATHER_UPDATE_INTERVAL (15 * 60 * 1000) // 15 хв -
інтервал оновлення погоди



#define NTP_TIMEOUT 30
// Таймаут синхронізації NTP (с)

const char *KYIV_TIMEZONE = "EET-2EEST,M3.5.0/3,M10.5.0/4";
// Часовий пояс Києва



#define BUTTON_PIN 3 // Пін кнопки (RX/GPIO3)



// --- Глобальні об'єкти ---

Adafruit_SSD1306 display(OLED_SCREEN_WIDTH,
OLED_SCREEN_HEIGHT, &Wire, OLED_RESET_PIN);

ESP8266WebServer server(80);



// --- Глобальні змінні ---

ConfigData currentConfig;                    // Поточна
конфігурація

WeatherData weatherData;                     // Дані
погоди

bool isAPMode = false;                       // Режим
точки доступу?
```

```
bool configSavedViaWeb = false;                // Конфігурацію збережено через веб?

bool timeIsSynced = false;                     // Час синхронізовано?

unsigned long lastNtpAttemptTime = 0;          // Остання спроба NTP

bool weatherDataValid = false;                 // Дані погоди валідні?

unsigned long lastWeatherUpdateTimestamp = 0; // Останнє оновлення погоди

unsigned long bootTimeMillis = 0;              // Час завантаження


byte currentScreen = 0;      // Поточний екран OLED

const byte totalScreens = 6; // Кількість екранів


unsigned long lastButtonCheckTime = 0;     // Остання перевірка кнопки

bool serialWasTemporarilyDisabled = false; // Serial тимчасово вимкнено?


// --- Функції EEPROM ---

// Зберігає конфігурацію в EEPROM

void saveConfiguration()

{

  Serial.println(F("Saving configuration..."));

  currentConfig.magicByte = CONFIG_MAGIC_BYTE; // "Магічний байт"


  EEPROM.begin(EEPROM_SIZE);

  EEPROM.put(EEPROM_CONFIG_START_ADDR, currentConfig);
```

```
  bool committed = EEPROM.commit();

  EEPROM.end();


  Serial.println(committed ? F("Config saved.") : F("ERROR:
Config save failed."));

}


// Завантажує конфігурацію з EEPROM. Повертає true, якщо
успішно.

bool loadConfiguration()

{

  Serial.println(F("Loading configuration..."));

  EEPROM.begin(EEPROM_SIZE);

  EEPROM.get(EEPROM_CONFIG_START_ADDR, currentConfig);

  EEPROM.end();


  if (currentConfig.magicByte == CONFIG_MAGIC_BYTE)

  {

    Serial.println(F("Valid config found."));

    // Забезпечення нуль-термінації

    currentConfig.ssid[sizeof(currentConfig.ssid) - 1] =
'\0';

    currentConfig.password[sizeof(currentConfig.password) -
1] = '\0';

    if (strlen(currentConfig.timezone) == 0)

    {

      strcpy(currentConfig.timezone, KYIV_TIMEZONE); //
Часовий пояс за замовчуванням

    }
```

```
    return true;
  }
  else
  {
    Serial.println(F("No valid config. Using defaults."));
    strcpy(currentConfig.ssid, "");
    strcpy(currentConfig.password, "");
    strcpy(currentConfig.timezone, KYIV_TIMEZONE);
    currentConfig.magicByte = 0;
    return false;
  }
}

// Скидає конфігурацію в EEPROM
void resetConfiguration()
{
  Serial.println(F("Resetting configuration..."));
  memset(&currentConfig, 0, sizeof(ConfigData));
  strcpy(currentConfig.timezone, KYIV_TIMEZONE);
  currentConfig.magicByte = 0;

  EEPROM.begin(EEPROM_SIZE);
  EEPROM.put(EEPROM_CONFIG_START_ADDR, currentConfig);
  bool committed = EEPROM.commit();
  EEPROM.end();
  Serial.println(committed ? F("Config reset.") : F("EEPROM
reset commit failed."));
```

```
}


// --- Функції WiFi та мережі ---
// Налаштування режиму точки доступу (AP)
void setupAPMode()
{
  Serial.println(F("Setting up AP mode..."));
  WiFi.mode(WIFI_AP);
  WiFi.softAP(AP_SSID, AP_PASSWORD);
  IPAddress apIP = WiFi.softAPIP();
  Serial.print(F("AP SSID: "));
  Serial.println(AP_SSID);
  Serial.print(F("AP IP: "));
  Serial.println(apIP);

  isAPMode = true;
  setupServerRoutesWeb(server, saveConfiguration,
resetConfiguration);
  displayAPModeScreenOled(display, AP_SSID,
apIP.toString().c_str());
}


// Підключення до Wi-Fi. Повертає true, якщо успішно.
bool connectToWiFi()
{
  if (strlen(currentConfig.ssid) == 0)
  {
```

```
    Serial.println(F("SSID not set."));
    return false;
  }
  Serial.print(F("Connecting to WiFi: "));
  Serial.println(currentConfig.ssid);

  WiFi.mode(WIFI_STA);
  WiFi.hostname("SmartClock-ESP01");
  WiFi.begin(currentConfig.ssid, currentConfig.password);

  displayConnectingScreenOled(display, currentConfig.ssid);

  unsigned long connectStartTime = millis();
  int dots_anim = 0;
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
    displayConnectingScreenOled(display,
currentConfig.ssid, dots_anim);
    dots_anim = (dots_anim + 1) % 4;

    if (millis() - connectStartTime > (WIFI_CONNECT_TIMEOUT
* 1000UL))
    {
      Serial.println(F("\nWiFi connect timeout."));
      WiFi.disconnect(true);
```

```
      return false;
    }
  }

  Serial.println(F("\nWiFi connected!"));
  Serial.print(F("IP: "));
  Serial.println(WiFi.localIP());
  Serial.print(F("RSSI: "));
  Serial.print(WiFi.RSSI());
  Serial.println(F(" dBm"));

  if (MDNS.begin("smartclock"))
  {
    Serial.println(F("MDNS: http://smartclock.local"));
    MDNS.addService("http", "tcp", 80);
  }
  else
  {
    Serial.println(F("MDNS setup error!"));
  }

  isAPMode = false;
  setupServerRoutesWeb(server, saveConfiguration,
resetConfiguration);
  return true;
}
```

```
// Синхронізація часу з NTP
void syncTimeWithNTP()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    Serial.println(F("NTP: WiFi not connected."));
    timeIsSynced = false;
    return;
  }
  Serial.print(F("NTP: Syncing time, TZ: "));
  Serial.println(currentConfig.timezone);

  configTime(currentConfig.timezone, "pool.ntp.org",
"time.nist.gov", "time.google.com");

  time_t now_epoch = time(nullptr);
  int attempts = 0;
  Serial.print(F("NTP: Waiting for sync"));
  while (now_epoch < 100000 && attempts < NTP_TIMEOUT)
  { // Чекаємо валідного часу
    delay(1000);
    Serial.print(".");
    now_epoch = time(nullptr);
    attempts++;
  }
  Serial.println();
```

```
  if (now_epoch > 100000)
  {
    struct tm timeinfo;
    localtime_r(&now_epoch, &timeinfo);
    Serial.print(F("NTP: Time synced: "));
    Serial.print(asctime(&timeinfo));
    timeIsSynced = true;
  }
  else
  {
    Serial.println(F("NTP: Failed to sync time."));
    timeIsSynced = false;
  }
  lastNtpAttemptTime = millis();
}

// --- Оновлення погоди ---
void updateWeatherData()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    Serial.println(F("Weather: WiFi not connected."));
    weatherDataValid = false;
    return;
  }
  if (!timeIsSynced)
```

```
  {
    Serial.println(F("Weather: Time not synced."));
    return;
  }
  if (String(WEATHER_API_KEY) == "YOUR_WEATHER_API_KEY" ||
strlen(WEATHER_API_KEY) < 10)
  {
    Serial.println(F("Weather: API Key not set."));
    weatherDataValid = false;
    strcpy(weatherData.condition_text, "Set API Key!");
    return;
  }

  Serial.println(F("Weather: Updating data..."));
  WiFiClient client;
  HTTPClient http;

  weatherApiUrl =
"http://api.weatherapi.com/v1/current.json?key=" +
String(WEATHER_API_KEY) + "&q=" +
String(WEATHER_API_LOCATION) + "&aqi=yes";

  http.begin(client, weatherApiUrl);
  http.setTimeout(15000); // Таймаут 15с
  int httpCode = http.GET();

  if (httpCode > 0)
  {
```

```
    if (httpCode == HTTP_CODE_OK || httpCode ==
HTTP_CODE_MOVED_PERMANENTLY)

    {

      String payload = http.getString();

      DynamicJsonDocument doc(2048); // Розмір JSON
підібрано

      DeserializationError error = deserializeJson(doc,
payload);



      if (error)

      {

        Serial.print(F("Weather: JSON parse failed: "));

        Serial.println(error.c_str());

        weatherDataValid = false;

      }

      else

      {

        JsonObject current = doc["current"];

        weatherData.temp_c = current["temp_c"].as<float>();

        weatherData.feelslike_c =
current["feelslike_c"].as<float>();



        const char *cond_text =
current["condition"]["text"];

        strncpy(weatherData.condition_text, cond_text ?
cond_text : "", sizeof(weatherData.condition_text) - 1);


weatherData.condition_text[sizeof(weatherData.condition_tex
t) - 1] = '\0';
```

```
        weatherData.condition_code =
current["condition"]["code"].as<int>();

        weatherData.is_day = current["is_day"].as<int>();

        weatherData.wind_kph =
current["wind_kph"].as<float>();

        weatherData.wind_degree =
current["wind_degree"].as<int>();



        const char *wind_d = current["wind_dir"];

        strncpy(weatherData.wind_dir, wind_d ? wind_d : "",
sizeof(weatherData.wind_dir) - 1);

        weatherData.wind_dir[sizeof(weatherData.wind_dir) -
1] = '\0';



        weatherData.pressure_mb =
current["pressure_mb"].as<float>();

        weatherData.humidity =
current["humidity"].as<float>();

        weatherData.precip_mm =
current["precip_mm"].as<float>();

        weatherData.cloud = current["cloud"].as<int>();



        if (current.containsKey("air_quality"))

        {

          JsonObject air_quality = current["air_quality"];

          weatherData.pm2_5 =
air_quality["pm2_5"].as<float>();

          weatherData.pm10 =
air_quality["pm10"].as<float>();

        }

        else
```

```
        {
          weatherData.pm2_5 = -1.0;
          weatherData.pm10 = -1.0;
        }

        weatherDataValid = true;
        lastWeatherUpdateTimestamp = millis();
        Serial.println(F("Weather: Data updated."));
      }
    }
    else
    {
      Serial.print(F("Weather: HTTP GET error, code: "));
      Serial.println(httpCode);
      weatherDataValid = false;
    }
  }
  else
  {
    Serial.print(F("Weather: HTTP GET failed: "));
    Serial.println(http.errorToString(httpCode).c_str());
    weatherDataValid = false;
  }
  http.end();
}
```

```
// --- Обробка кнопки ---

// Перевіряє натискання кнопки. Повертає true, якщо
натиснуто.

bool checkButtonPress()

{

  if (millis() - lastButtonCheckTime < 300)

  { // Антидребезг 300 мс

    return false;

  }

  lastButtonCheckTime = millis();


  // Пін RX (GPIO3) для кнопки. Serial може заважати.

  serialWasTemporarilyDisabled = Serial;

  if (serialWasTemporarilyDisabled)

  {

    Serial.flush();

    Serial.end();

    delay(10);

  }


  pinMode(BUTTON_PIN, INPUT_PULLUP);

  delay(5);

  bool pressed = (digitalRead(BUTTON_PIN) == LOW);


  if (serialWasTemporarilyDisabled)

  {

    Serial.begin(115200);
```

```
  }
  return pressed;
}

// --- Оновлення екрану ---
void updateDisplayScreens()
{
  static unsigned long lastDisplayUpdateTime = 0;
  bool forceUpdateOled = false;

  if (checkButtonPress())
  {
    currentScreen = (currentScreen + 1) % totalScreens; //
Наступний екран
    Serial.print(F("Button. Screen: "));
    Serial.println(currentScreen);
    forceUpdateOled = true;
  }

  if (forceUpdateOled || (millis() - lastDisplayUpdateTime
> 1000))
  { // Оновлення кожну секунду або примусово
    lastDisplayUpdateTime = millis();
    if (isAPMode)
    {
      if (forceUpdateOled)
        displayAPModeScreenOled(display, AP_SSID,
WiFi.softAPIP().toString().c_str());
```

```
    }

    else if (WiFi.status() == WL_CONNECTED)

    {

      displayStatusScreenOled(display, currentScreen,
totalScreens, timeIsSynced, weatherData, weatherDataValid,
bootTimeMillis);

    }

    else

    {

      if (forceUpdateOled)

        displayConnectingScreenOled(display,
currentConfig.ssid);

    }

  }

}



// --- Setup ---

void setup()

{

  Serial.begin(115200);

  unsigned long serialStartTime = millis();

  while (!Serial && (millis() - serialStartTime < 2000))

  {

    delay(10);

  } // Чекаємо Serial (2с)

  Serial.println(F("\n\n===== SmartClock v3.0.2 Booting
====="));
```

```
  bootTimeMillis = millis();


  initOled(display, OLED_SDA_PIN, OLED_SCL_PIN);

  displayBootScreenOled(display, "SmartClock", "v3.0.2
Loading...");


  EEPROM.begin(EEPROM_SIZE);

  if (!loadConfiguration())

  {

    saveConfiguration(); // Зберігаємо дефолтну, якщо немає
валідної

  }


  if (strlen(currentConfig.ssid) > 0 && connectToWiFi())

  {

    isAPMode = false;

    syncTimeWithNTP();

    if (timeIsSynced)

    {

      updateWeatherData();

    }

  }

  else

  {

    Serial.println(F("Connect failed or no SSID. Starting
AP mode."));

    setupAPMode();

  }
```

```
  currentScreen = 0;
  updateDisplayScreens();
}

// --- Loop ---
void loop()
{
  server.handleClient();

  if (!isAPMode && WiFi.status() == WL_CONNECTED)
  {
    MDNS.update();
  }

  updateDisplayScreens(); // Оновлення OLED (кнопка всередині)

  if (configSavedViaWeb)
  {
    Serial.println(F("Config saved via web. Restarting in 5s..."));
    displaySystemMessageOled(display, "Settings Saved", "Restarting...");

    unsigned long restartInitiateTime = millis();
    while (millis() - restartInitiateTime < 5000)
    { // Чекаємо 5с
```

```
      server.handleClient();
      delay(10);
    }
    ESP.restart();
  }

  // Періодична синхронізація часу (кожні 6 годин)
  if (!isAPMode && WiFi.status() == WL_CONNECTED)
  {
    if (!timeIsSynced || (millis() - lastNtpAttemptTime >
(6 * 60 * 60 * 1000UL)))
    {
      syncTimeWithNTP();
    }
  }
  else
  {
    timeIsSynced = false;
  }

  // Періодичне оновлення погоди
  if (!isAPMode && WiFi.status() == WL_CONNECTED &&
timeIsSynced)
  {
    if (!weatherDataValid || (millis() -
lastWeatherUpdateTimestamp >= WEATHER_UPDATE_INTERVAL))
    {
```

```
      if (String(WEATHER_API_KEY) != "YOUR_WEATHER_API_KEY"
&& strlen(WEATHER_API_KEY) > 10)

      {

        updateWeatherData();

      }

      else if (!weatherDataValid)

      {

        strcpy(weatherData.condition_text, "Set API Key!");

      }

    }

  }


  // Перевірка Wi-Fi та перепідключення

  static unsigned long lastWifiCheckTime = 0;

  if (!isAPMode && (millis() - lastWifiCheckTime >
(WIFI_RECONNECT_INTERVAL * 1000UL)))

  {

    lastWifiCheckTime = millis();

    if (WiFi.status() != WL_CONNECTED)

    {

      Serial.println(F("WiFi lost. Reconnecting..."));

      timeIsSynced = false;

      weatherDataValid = false;

      displayConnectingScreenOled(display,
currentConfig.ssid, 0, true); // Екран перепідключення

      if (!connectToWiFi())

      {

        Serial.println(F("Reconnect failed."));
```

```
      }
      else
      {
        syncTimeWithNTP();
        if (timeIsSynced)
          updateWeatherData();
      }
    }
  }
  yield(); // Для ESP8266
}
```

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
Михайла ДЕРЕВ’ЯГИ
(ім’я, прізвище)

Тематика створення інтелектуальних IoT-пристроїв, зокрема персональних інформаційних систем, є надзвичайно актуальною у контексті розвитку концепції "розумного дому", мобільної автоматизації та інтеграції хмарних сервісів. Використання мікроконтролера ESP-01 як основи для реалізації годинника з функцією прогнозу погоди є технічно виправданим і відповідає сучасним тенденціям проєктування енергоефективних компактних пристроїв.

Кваліфікаційна робота повністю відповідає затвердженій темі. Усі поставлені завдання реалізовані успішно: здійснено вибір апаратної бази, обґрунтовано вибір сервісу WeatherAPI, реалізовано алгоритм обробки метеоданих, створено веб-інтерфейс налаштування та виконано тестування пристрою в реальних умовах.

У результаті створено повнофункціональний прототип розумного годинника, здатного відображати час, прогноз погоди та додаткові параметри навколишнього середовища у режимі реального часу. Веб-інтерфейс дозволяє користувачеві налаштовувати пристрій без потреби в перепрошивці, що значно покращує зручність експлуатації.

Пояснювальна записка оформлена згідно з вимогами ДСТУ. Структура роботи логічна, матеріал викладено послідовно. Особливо слід відзначити якісне подання апаратної схеми, програмної логіки та принципів взаємодії пристрою з мережею Wi-Fi.

Розроблений пристрій має потенціал для практичного використання як елемент персональної електроніки або складова системи розумного середовища. Проєкт демонструє реальні навички мікроконтролерного програмування, роботи з периферією, мережевими протоколами та базової веб-розробки.

Кваліфікаційна робота виконана на високому рівні.

Робота заслуговує на оцінку “відмінно”.

Рецензент *викладач*
(науковий ступінь, посада)

«30» 05 2025 р. _____ Сергій РУДИЙ
(підпис) (ім’я, прізвище)

З рецензією ознайомлений _____ Михайло ДЕРЕВ’ЯГА
(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

Михайло ДЕРЕВ`ЯГА
(ім'я, прізвище)

1. *Кваліфікаційна робота на тему «Розумний годинник на базі ESP-01 з функцією*
2. *Метою кваліфікаційної роботи є розробка пристрою на базі ESP-01, що здатен відображати актуальну інформацію про погоду за допомогою підключення до інтернету..*
3. *Кваліфікаційна робота відповідає темі, затвердженій наказом начальника коледжу..*
4. *Кваліфікаційна робота виконана здобувачем освіти самостійно.*
5. *Здобувач освіти показав високі вміння роботи з літературними джерелами, аналізу теоретичного та практичного матеріалу, прийняття обґрунтованих технічних рішень, а також застосування сучасних апаратно-програмних засобів..*
6. *Михайло ДЕРЕВ'ЯГА показав достатній рівень дотримання вимог державних стандартів при виконанні кваліфікаційної роботи в цілому та оформленні пояснювальної записки.*
7. *Рівень виконаної кваліфікаційної роботи заслуговує оцінку «добре», відповідає набутих випускником знань, умінь та навичок, вимогам освітньої характеристики фахівця і можливість присвоєння йому кваліфікації фахівця освітнього ступеня «Бакалавр» спеціальності 123 «Комп'ютерна інженерія».*

Керівник кваліфікаційної роботи

«10» 06 2025 р. _____ Галина ДАНИЛІНА
(підпис) (ім'я, прізвище)