МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут комп’ютерних технологій, автоматики та метрології

Кафедра "Комп'ютеризовані системи автоматики"

ЗВІТ

про виконання **курсової роботи**

з навчальної дисципліни**: «»,**

на тему: «Система контролю показників навколишнього серидовища»

Виконала:

студентка групи ІР-

Прийняв:

к.н.т, доцент

Яцук Ю. В.

Львів – 2025

ЗМІСТ

МЕТА ТА ЗАВДАННЯ НА ПРОЕКТ

Метою проєкту є створення системи моніторингу навколишнього середовища, яка забезпечує вимірювання параметрів температури, вологості, атмосферного тиску та рівня освітленості в реальному часі. Система побудована на основі мікроконтролера ESP32-WROOM-32D, використовує датчик BME280 для збору метеорологічних даних, фоторезистор для вимірювання яскравості, і передає інформацію на комп'ютер через протокол MQTT, де дані зберігаються у базі даних SQLite та відображаються через вебінтерфейс на Flask.

Завдання проекту:

1. **Апаратне забезпечення:**
   - Мікроконтролер: ESP32-WROOM-32D
   - Датчики:
     - Температура, вологість, тиск: BME280
     - Яскравість: фоторезистор (підключений до аналогового входу ESP32)
   - Живлення: USB Type C або зовнішнє джерело живлення (5V)
   - Комунікація: Wi-Fi-модуль, вбудований в ESP32
2. **Програмне забезпечення мікроконтролера:**
   - Мова програмування: Arduino (ESP32 Core)
   - Комунікаційний протокол: MQTT (для надсилання даних на сервер)
3. **База даних:**
   - Платформа: SQLite (локально на комп'ютері)
   - Формат даних: JSON-повідомлення, отримані через MQTT
   - Функціонал: зберігання, перегляд і подальший аналіз екологічних параметрів

4. **Backend-сервер (на комп’ютері):**
    - Мова програмування: Python
    - Бібліотеки: paho-mqtt (для приймання MQTT повідомлень), sqlite3 (для роботи з базою даних)
    - Призначення: приймання даних з ESP32 через MQTT, зберігання їх у SQLite, забезпечення доступу до даних через веб-сервер
5. **Frontend (вебінтерфейс):**
    - Технології: HTML, CSS, Flask (на Python)
    - Розташування коду: комп’ютер (локальний веб-сервер)
    - Функціонал: візуалізація зібраних даних користувачу в зручному графічному інтерфейсі через браузер

ВСТУП

Актуальність

Контроль вологості повітря є важливим аспектом для багатьох сфер, таких як сільське господарство, зберігання продукції, охорона здоров’я, а також забезпечення комфортних побутових умов. Вологість повітря безпосередньо впливає на якість зберігання товарів, рівень комфорту, стан здоров’я людей і навіть на енергоспоживання в приміщеннях. Наприклад, у теплицях підтримання оптимального рівня вологості сприяє здоровому росту рослин, а в медичних закладах допомагає запобігати поширенню інфекцій та створює сприятливі умови для пацієнтів.

Надмірна вологість може спричинити розвиток грибків, плісняви та інших шкідливих мікроорганізмів, тоді як її нестача призводить до сухості шкіри, подразнення слизових оболонок і загального погіршення самопочуття. З огляду на це, забезпечення належного рівня вологості є критично важливим для створення здорового та безпечного середовища у різних умовах.

Мета проекту

Метою проєкту є розробка інтегрованої системи контролю вологості повітря, яка дозволяє здійснювати моніторинг та візуалізацію даних у режимі реального часу. Система має бути простою у використанні, надійною, адаптивною до різних умов експлуатації та здатною передавати дані на хмарну платформу для їхнього зберігання й аналізу. Збір показників вологості буде відбуватися з заданим інтервалом (наприклад, кожні 10 хвилин), а зібрані дані передаватимуться через MQTT на базу даних, де вони будуть доступні для обробки та подальшого використання.

Практична цінність проекту

Розроблена система може стати ефективним інструментом для моніторингу вологості в різноманітних умовах. Серед основних сфер застосування:

1. **Побутові умови** — підтримання комфортного мікроклімату для мешканців, що позитивно впливає на здоров’я та зберігає цілісність меблів, електроніки та інших предметів інтер’єру.
2. **Тепличне господарство** — створення оптимальних умов для вирощування рослин, що підвищує врожайність і якість продукції.
3. **Склади та логістика** — контроль вологості задля збереження якості товарів, особливо продуктів харчування, текстилю та електроніки.
4. **Медичні заклади** — забезпечення відповідного мікроклімату для пацієнтів і медичного обладнання, що сприяє профілактиці інфекцій та прискорює одужання.
5. **Офіси та комерційні приміщення** — покращення умов праці, що сприяє підвищенню продуктивності персоналу та зменшенню рівня захворюваності.

Веб-інтерфейс дозволяє не лише отримувати актуальні показники вологості, але й змінювати параметри роботи пристрою відповідно до потреб користувача. Рішення може масштабуватися та легко інтегруватися в більші екосистеми розумного дому або промислового моніторингу.

РОЗДІЛ 1. АНАЛІЗ ПИТАННЯ

Для створення курсового проєкту я використовую наступні компоненти:

1) **Мікроконтролер ESP32-WROOM-32D**

- **Опис:** ESP32-WROOM-32D — це модуль від Espressif, що поєднує в собі двоядерний мікроконтролер з підтримкою Wi-Fi та Bluetooth. Завдяки високій продуктивності, широкому набору периферії та підтримці бездротових технологій, ESP32 є ідеальним вибором для IoT-проєктів, включаючи сенсорні системи та системи віддаленого моніторингу.
- **Основні характеристики:**
  - Процесор: Два ядра Xtensa® 32-біт LX6
  - Тактова частота: до 240 МГц
  - ОЗП: 520 КБ SRAM
  - Флеш-пам’ять: до 4 МБ (залежно від модуля)
  - Підключення: Wi-Fi 802.11 b/g/n та Bluetooth 4.2 (Classic + BLE)
  - Інтерфейси: GPIO, SPI, I2C, UART, PWM, ADC, DAC, I2S тощо
  - Живлення: 3.3V
  - Робоча температура: -40°C до +85°C
- **Переваги:**
  - Підтримка бездротових протоколів (Wi-Fi, Bluetooth)
  - Висока продуктивність для паралельної обробки даних
  - Велика кількість доступних входів/виходів
  - Розширена підтримка бібліотек в Arduino, PlatformIO та ESP-IDF
- **Бібліотеки для використання:**
  - WiFi.h – для підключення до бездротової мережі
  - PubSubClient.h – для зв’язку через MQTT-протокол
  - Adafruit_BME280.h – для роботи з BME280
  - Wire.h – для управління завданнями та сенсорами

2) **Датчик температури, вологості та тиску BME280**

- **Опис:** BME280 — це цифровий сенсор від Bosch, який дозволяє вимірювати температуру, вологість та атмосферний тиск. Він відзначається високою точністю та стабільністю вимірювань.
- **Основні характеристики:**
  - Діапазон температур: від -40°C до +85°C (точність ±1.0°C)
  - Діапазон вологості: 0–100% RH (точність ±3% RH)
  - Діапазон тиску: 300 – 1100 hPa (точність ±1 hPa)
  - Інтерфейс: I2C
  - Живлення: 1.8V – 3.6V
- **Переваги:**
  - Три параметри в одному модулі
  - Компактність та надійність
  - Легка інтеграція через I2C та підтримка бібліотек Arduino/PlatformIO

3) **Фоторезистор**

- **Опис:** Фоторезистор — це резистор, що змінює свій опір залежно від рівня освітленості. Зчитування яскравості оточення здійснюється через аналоговий вхід ESP32.
- **Основні характеристики:**
  - Тип сигналу: аналоговий
  - Робоча напруга: 3.3V
  - Чутливість: змінюється від ~10 кОм до >1 МОм залежно від освітлення
  - Використовується в парі з резистором у подільнику напруги
- **Переваги:**
  - Простота конструкції та використання
  - Швидка реакція на зміну яскравості
  - Дешева альтернатива цифровим сенсорам світла

4) Програмна частина

- **MQTT-зв’язок:**

  ESP32 надсилає дані з датчиків через MQTT-брокер на комп’ютер. Передача відбувається по локальній мережі.

- **Python (серверна частина):**

  На ПК працює Python-скрипт, який отримує дані по MQTT та зберігає їх у базу даних SQLite. Для цього використовуються бібліотеки paho-mqtt, sqlite3.

- **Flask (веб інтерфейс):**

  Локальний сайт на Flask дозволяє переглядати збережені дані в зручному вигляді. Можлива також фільтрація та сортування по часу.

РОЗДІЛ 2. РОЗРОБЛЕННЯ СИСТЕМИ КОНТРОЛЮ ПОКАЗНИКІВ НАВКОЛИШНЬОГО СЕРИДОВИЩА

2.1 ФУНКЦІОНАЛЬНА СХЕМА

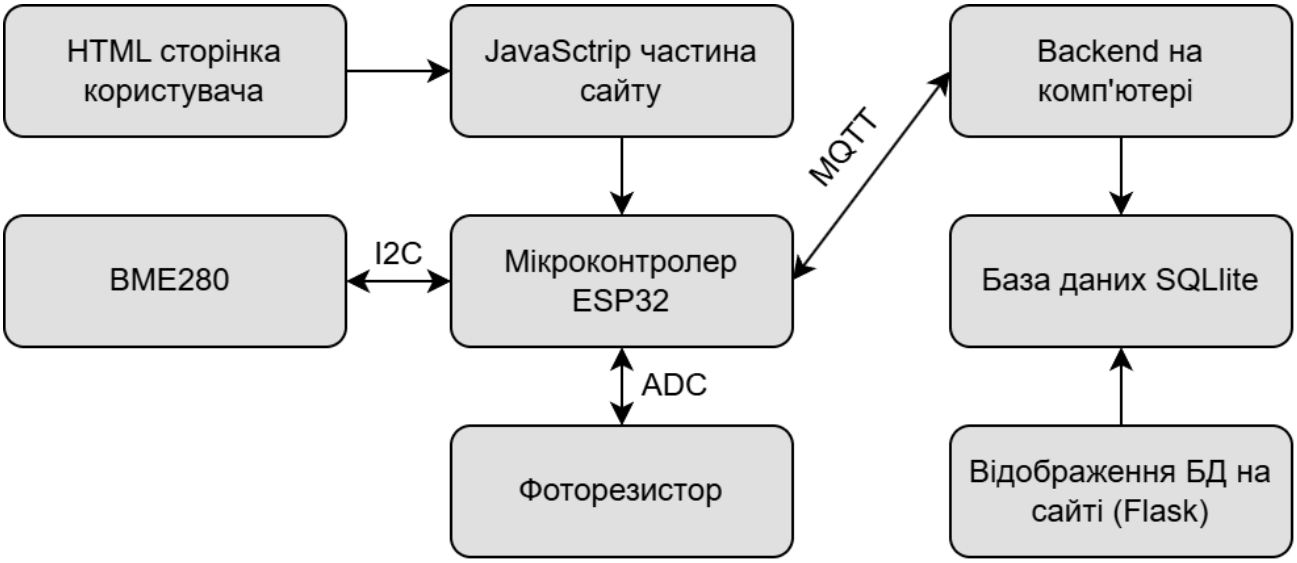


Рис. 1. Функціональна схема

1. HTML сторінка користувача

Це інтерфейс взаємодії кінцевого користувача з системою. HTML-сторінка відображає інформацію, отриману з бази даних, у зручному вигляді через браузер. Вона є частиною вебзастосунку, реалізованого на Flask.

2. JavaScript частина сайту

JavaScript забезпечує динамічність HTML-сторінки. У даному проєкті він може використовуватись для ініціалізації запитів до сервера Flask, оновлення даних без перезавантаження сторінки (через AJAX), а також для побудови графіків або таблиць.

3. Мікроконтролер ESP32

ESP32 є центральним вузлом системи, який збирає дані з підключених сенсорів (BME280 та фоторезистор), обробляє їх та передає на комп’ютер через протокол MQTT.

- Підключення до BME280 здійснюється через інтерфейс I2C.
- Значення з фоторезистора зчитуються через вбудований ADC (аналогово-цифровий перетворювач).
- Після збору даних ESP32 формує MQTT-повідомлення та надсилає його до брокера.

4. BME280

Цифровий сенсор вимірювання температури, вологості та атмосферного тиску. Він підключений до ESP32 через інтерфейс I2C. Завдяки своїй точності та надійності, BME280 забезпечує якісні дані для подальшого аналізу.

5. Фоторезистор

Аналоговий сенсор освітленості. Його опір змінюється залежно від рівня освітлення, що дозволяє мікроконтролеру виміряти інтенсивність світла за допомогою вбудованого АЦП (ADC). Результат вимірювання також включається до MQTT-повідомлення.

6. Backend на комп’ютері

Програмна частина, що отримує MQTT-повідомлення від ESP32. Скрипт на Python (з використанням бібліотеки paho-mqtt) приймає ці дані, парсить їх та зберігає в базу даних SQLite для подальшої обробки.

7. База даних SQLite

Локальна реляційна база даних, в якій зберігаються отримані значення з сенсорів. Вона є джерелом даних для виводу на вебсторінці. SQLite обрана за простоту інтеграції та відсутність потреби у встановленні серверної СУБД.

8. Відображення БД на сайті (Flask)

Фреймворк Flask використовується для створення вебсайту, який надає доступ до збережених даних у базі. Дані з SQLite виводяться у вигляді таблиць, графіків або звітів на HTML-сторінці. Це дозволяє користувачеві аналізувати стан довкілля в реальному часі або в ретроспективі.

2.2 ПРИНЦИПОВА СХЕМА

Компоненти схеми:

1. **ESP32-DEVKITC-32D** — головний мікроконтролер, що обробляє сигнали з датчиків і передає їх до комп’ютера через бездротовий зв’язок.
2. **BME280** — цифровий сенсор для вимірювання температури, вологості та атмосферного тиску.
3. **Фоторезистор + резистор 10кОм** — аналоговий сенсор освітленості, підключений як подільник напруги.

З'єднання компонентів:

ESP32-DEVKITC-32D:

Живлення:

- PIN **3V3** (вивід 1) ESP32 підключений до **VIN** модуля BME280 (вивід 1).
- PIN **GND** (вивід 2) ESP32 підключений до **GND** модуля BME280 (вивід 2) та до загальної землі всіх компонентів.

BME280 (U2):

- **VIN** — підключено до 3.3V ESP32.
- **GND** — підключено до GND ESP32.
- **SCL (вивід 3)** — підключено до GPIO **22** ESP32 (вивід 14).
- **SDA (вивід 4)** — підключено до GPIO **21** ESP32 (вивід 11).
- Комунікація відбувається через інтерфейс **I2C**.

Фоторезистор (U3) + резистор R1:

- Один кінець **фоторезистора** підключений до **3.3V (VIN)**.
- Інший кінець фоторезистора підключений до:
  - Аналогового входу **GPIO34 (D34)** ESP32.
  - Одного кінця **резистора 10 кОм (R1)**.
- Інший кінець резистора R1 підключено до **GND**.
- Ця конфігурація утворює **подільник напруги**, де значення напруги на GPIO34 залежить від освітленості.

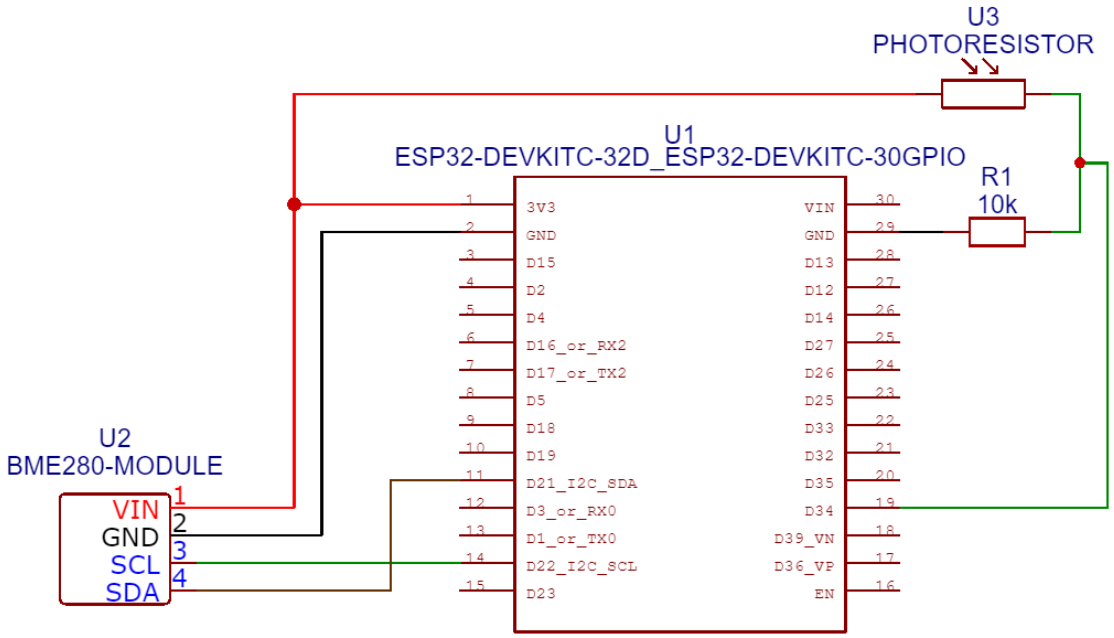


Рис. 2. Принципова схема

Опис роботи:

1. **Збір даних з датчика DHT22:**

   Датчик DHT22 передає дані про вологу та температуру на ESP8266 через аналоговий пін D3.

2. **Керування кольором RGB модуля:**

   ESP8266 керує RGB світлодіодним модулем через цифрові піни GPIO. Три піни (D6, D7, D8) використовуються для керування червоним, зеленим та синім каналами відповідно.

   Використовується широтно-імпульсна модуляція (PWM) для регулювання яскравості кожного кольору.

2.3 ПРОТОКОЛ ПЕРЕДАЧІ ДАНИХ

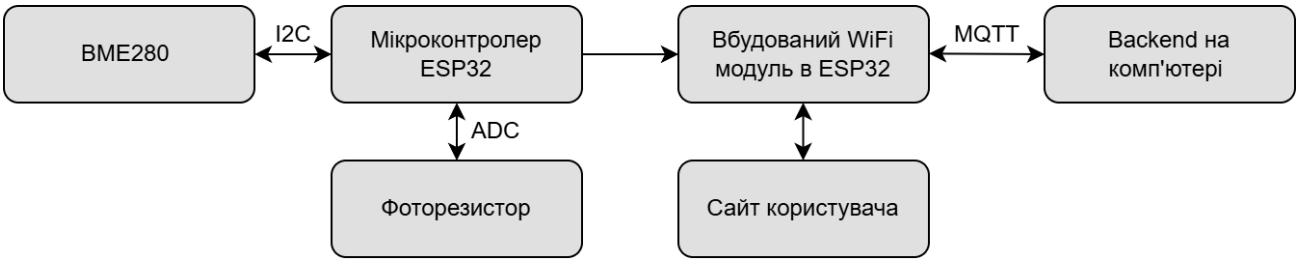


Рис. 3. Схема передачі даних

Опис протоколу передачі даних для проекту

BME280 (Сенсор температури, вологості та тиску):

- **Дані:** Температура, вологість, атмосферний тиск.
- **Інтерфейс:** Цифровий інтерфейс I²C.
- **Протокол:** I²C (послідовна шина з двома проводами: SCL та SDA).
  - **SDA (дані):** Передає/отримує дані.
  - **SCL (тактова лінія):** Синхронізує обмін.

Фоторезистор:

- **Дані:** Аналоговий рівень освітлення.
- **Інтерфейс:** Аналоговий вхід ESP32 (ADC).
- **Протокол:** Напруга у подільнику зчитується через **вбудований АЦП** ESP32 (ADC).
    - Напруга змінюється в залежності від освітлення, обробляється у вигляді значень від 0 до 4095.

ESP32 (Мікроконтролер):

- **Збір даних:**
    - Зчитує **цифрові значення** з BME280 через I²C.
    - Зчитує **аналоговий сигнал** від фоторезистора через вбудований АЦП.
- **Обробка:** Формує структуровані дані (JSON).
- **Передача даних:**
    - Відправляє зібрані дані через вбудований Wi-Fi модуль.

Wi-Fi модуль (вбудований в ESP32):

- **Дані:** Температура, вологість, тиск, освітленість.
- **Інтерфейс:** Wi-Fi (IEEE 802.11 b/g/n).
- **Протокол:**
    - **MQTT:** Для легкого обміну повідомленнями з брокером або сервером (легкий протокол для IoT).
    - **Або HTTP/HTTPS:** Для відправлення даних на вебсервер або REST API.

Backend:

- **Дані:** Отримує та зберігає дані з ESP32.

- **Інтерфейс:** MQTT-брокер або вебсервер.
- **Протокол:** MQTT.
    - Зберігання здійснюється у базі даних (SQLite).

Клієнтський веб-інтерфейс (вебсайт):

- **Дані:** Отримує та візуалізує дані з серверного API чи бази даних.
- **Інтерфейс:** Веб-браузер.
- **Протокол:** HTTP/HTTPS (REST API або WebSocket для реального часу).
    - Відображення графіків, таблиць та значень сенсорів у реальному часі.

2.4 ДИЗАЙН БД

ER (Entity-Relationship) діаграма представляє структуру бази даних. У нашому випадку використовується спрощена модель, оскільки для данного пристрою немає необхідності у базі даних з декількома табличками.

Для зберігання даних про температуру, вологість, тиск, освітлення, використовується одна таблиця sensor_data. Дизайн таблиці представлений на Рис. 3.

sensor_data		
id	Int	PK
timestamp	String	
temperature	Real	
brigthness	Real	
humidity	Real	
pressure	Real	

Рис. 3. Дизайн бази даних

Опис полів таблиці sensor_data**:**

- **id**: Первинний ключ (PRIMARY KEY). Автоматично зростаюче унікальне ціле число, що використовується для ідентифікації кожного запису в таблиці.
- **timestamp**: Текстове поле, яке зберігає часову мітку, а саме час, коли були зібрані дані.
- **temperature**: Поле типу REAL, що зберігає значення температури, отримане з сенсора BME280.
- **brightness**: Поле типу REAL, яке зберігає значення освітленості, отримане з фоторезистора (через АЦП ESP32).
- **humidity**: Поле типу REAL, що зберігає значення вологості з BME280.
- **pressure**: Поле типу REAL, що зберігає атмосферний тиск, також зчитується з BME280.

Вибір дизайну бази даних

Через те, що у цьому проєкті використовується **локальне зберігання** даних у **SQLite**, було обрано **єдину таблицю sensor_data**, що об'єднує всі показники в одному місці. Такий підхід спрощує реалізацію, підтримку та обробку даних.

2.5 FRONT-END

Фронтенд системи реалізовано у вигляді веб-інтерфейсу, що дозволяє користувачам переглядати поточні значення **температури**, **вологості**, **тиску** та **освітленості** в режимі реального часу. Також передбачено можливість керування параметрами системи. Веб-інтерфейс створено із застосуванням стандартних веб-технологій: **HTML**, **CSS** та **JavaScript**. Також реалізовано ще один сайт який відображає значення з бази даних.

Опис реалізації

1. **HTML**

   Визначає структуру веб-сторінки: блоки для відображення поточних значень сенсорів, заголовки, кнопки керування тощо. HTML формує основу взаємодії з користувачем.
2. **CSS**

   Забезпечує візуальне оформлення: кольорову схему, шрифти, розмітку, відступи та адаптивність. Це робить інтерфейс привабливим і зручним для користувача як на ПК, так і на мобільних пристроях.
3. **JavaScript**

   Відповідає за динамічну взаємодію з користувачем. Він обробляє запити до локальної бази даних або мікроконтролера, оновлює значення в інтерфейсі в реальному часі та реагує на події (натискання кнопок, введення даних тощо).

Причини вибору HTML, CSS та JavaScript

1. **Простота та доступність**

   Ці технології є базовими для веб-розробки, не потребують складного середовища, легко інтегруються з іншими компонентами системи (мікроконтролером або локальним сервером).

2. **Кросплатформеність**

   Фронтенд доступний з будь-якого пристрою, що має браузер — смартфон, планшет, ноутбук або ПК. Це гарантує універсальний доступ до системи моніторингу.
3. **Гнучкість та адаптивність**

   CSS дозволяє створити адаптивний інтерфейс, зручний на будь-якому екрані, а JavaScript забезпечує роботу без перезавантаження сторінки — дані оновлюються миттєво, створюючи ефект "живої" панелі моніторингу.

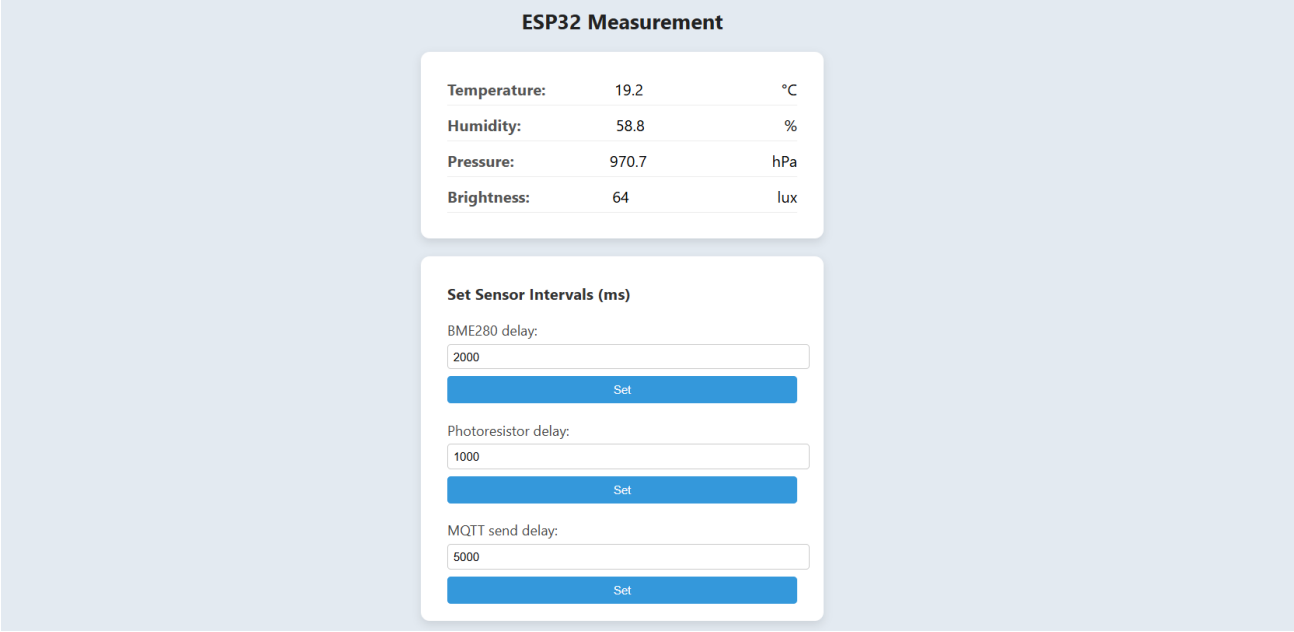


Рис. 4. Сайт що розміщується на ESP

На Рис. 4 можемо побачити сайт що розміщується на ESP, на ньому зображені показники температури, вологісті, тиску, освітлення, які є максимально актуальними оскільки єсп напряму передає дані на сайт, також тут є три поля вводу з кнопками, коже поле вводу дозволяє ввести затримку для якогось типу збирання чи надсилання даних. BME280 delay визначає період з яким будуть зніматись значення з сенсора BME280. Photoresistor delay визначає період з яким будуть зніматись значення з фоторезистора. MQTT delay визначає період з яким будуть надсилатись дані по MQTT.

Коротко код цього сайту з поясненням:

```
  <div class="card">
<div class="value">
  <span class="label">Temperature:</span>
  <span class="data" id="temp">--</span>
  <span>&deg;C</span>
</div>
<div class="value">
  <span class="label">Humidity:</span>
  <span class="data" id="hum">--</span>
  <span>%</span>
</div>
<div class="value">
  <span class="label">Pressure:</span>
```

```
  <span class="data" id="pres">--</span>
  <span>hPa</span>
</div>
<div class="value">
  <span class="label">Brightness:</span>
  <span class="data" id="lux">--</span>
  <span>lux</span>
</div>
```

- **4 рядки** з даними: температура, вологість, тиск, освітленість.
- Значення виводяться в **елементи з** `id="..."`, які оновлюються JavaScript'ом (у реальному часі).

```
<div class="card">
  <h3>Set Sensor Intervals (ms)</h3>

  <div class="setting-block">
    <label for="bme">BME280 delay:</label>
    <input type="number" id="bme">
    <button onclick="setBME()">Set</button>
  </div>

  <div class="setting-block">
    <label for="photo">Photoresistor delay:</label>
    <input type="number" id="photo">
    <button onclick="setPhoto()">Set</button>
  </div>

  <div class="setting-block">
    <label for="mqtt_delay">MQTT send delay:</label>
    <input type="number" id="mqtt_delay">
    <button onclick="setMQTT()">Set</button>
  </div>
</div>
```

- Містить **3 секції** для зміни затримки (інтервалу) оновлення:
  - **BME280** (температура, вологість, тиск)
  - **Фоторезистор** (освітленість)
  - **MQTT** (частота надсилання даних)
- Кнопки викликають JavaScript-функції: `setBME()`, `setPhoto()`, `setMQTT()`.

Рис. 5. Початкова сторінка з відображання даних БД

На Рис. 5 зображено початкову сторінку сайті що розміщується на комп’ютері та відображає вміст БД, на цій сторінці є кнопка яка при натисканні перекидає на сторінку на якій відображаються всі сенсори.

Коротко код цього сайту з поясненням:

```
@app.route("/")
def index():
    return render_template("index.html")
```

- Обробляє запит до кореневої сторінки (`/`).

- Повертає HTML-шаблон `index.html` — це веб-інтерфейс, який ти навів у попередньому повідомленні.

```
<!DOCTYPE html>
<html>
<head>
    <title>Sensor Dashboard</title>
</head>
<body>
    <h1>Welcome to Sensor Dashboard</h1>
    <p><a href="/data">View All Sensor Data</a></p>
</body>
</html>
```

- `a href="/data"`: Посилання на сторінку (`/data`), де виводиться таблиця з даними з БД.

All Sensor Data

ID	Timestamp	Temperature	Brightness	Humidity	Pressure
77	2025-05-14 03:20:54	22.9	54.0	47.0	975.3
76	2025-05-14 03:20:53	22.9	44.0	47.0	975.3
75	2025-05-14 03:20:52	23.0	56.0	46.8	975.3
74	2025-05-14 03:20:51	23.0	68.0	46.8	975.3
73	2025-05-14 03:20:50	23.1	70.0	46.7	975.3
72	2025-05-14 03:20:49	23.1	68.0	46.7	975.3
71	2025-05-14 03:20:48	23.2	68.0	46.8	975.3
70	2025-05-14 03:20:46	23.3	70.0	46.9	975.3
69	2025-05-14 03:20:45	23.3	70.0	46.9	975.3
68	2025-05-14 03:20:44	23.3	68.0	47.0	975.3
67	2025-05-14 03:20:43	23.3	68.0	47.0	975.3
66	2025-05-14 03:20:42	23.4	72.0	46.8	975.3
65	2025-05-14 03:20:41	23.4	71.0	46.8	975.3

Рис. 6. Відображення даних з БД

На Рис. 6 зображено сторінку яка відображає у вигляді таблички всі значення з БД, відсортовані від самих актуальних до самих неактуальних.

Коротко код цього сайту з поясненням:

```
@app.route("/data")
def data():
    sensor_data = get_all_data()
    return render_template("all_data.html", data=sensor_data)
```

- `@app.route("/data")`: Вказує, що ця функція виконується, коли користувач переходить на сторінку `/data`.

- `get_all_data()`: Функція, яка зчитує всі дані з бази даних (з SQLite).

- `render_template("all_data.html", data=sensor_data)`: Рендерить HTML-шаблон `all_data.html` і передає в нього змінну `data`, яка містить всі зчитані дані.

```
<table>
    <tr>
        <th>ID</th>
        <th>Timestamp</th>
        <th>Temperature</th>
        <th>Brightness</th>
        <th>Humidity</th>
        <th>Pressure</th>
    </tr>
```

- Заголовки стовпців для даних із бази.

```
    {% for row in data %}
        <tr>
            <td>{{ row[0] }}</td>
            <td>{{ row[1] }}</td>
            <td>{{ row[2] }}</td>
            <td>{{ row[3] }}</td>
            <td>{{ row[4] }}</td>
            <td>{{ row[5] }}</td>
        </tr>
        {% endfor %}
    </table>
```

- Цикл, який виводить кожен рядок з data, отриманого з Python-функції get_all_data().

2.6 BACK-END

Опис архітектури проекту

Архітектура проєкту складається з таких компонентів: вбудованого пристрою на основі ESP32**,** протоколу MQTT**,** сервера з Python та SQLite, а також локального веб-інтерфейсу на Flask.

1. Мікроконтролер (ESP32-WROOM-32D)

- Виконує зчитування параметрів навколишнього середовища:
  - **Температура, вологість і тиск** — за допомогою сенсора **BME280**.
  - **Яскравість оточення** — за допомогою **фоторезистора**.
- Підключений до Wi-Fi та працює як MQTT-клієнт.
- Передає зібрані дані через MQTT на сервер (комп’ютер).
- Публікує дані з фіксованою періодичністю на обраний топік.

2. Протокол обміну — MQTT

- MQTT використовується для надійного, легковагового обміну даними між ESP32 та комп’ютером.
- **Сервер на Python** є брокером та отримує повідомлення в реальному часі.

3. Серверна частина (Backend) на Python

- Отримує дані з ESP32 через MQTT.
- Обробляє отримані повідомлення та зберігає їх у **базу даних SQLite**.
- Має доступ до локального сховища даних та забезпечує їх передачу на веб-інтерфейс.

4. База даних (SQLite)

- Зберігає всі зчитані значення: **час**, **температуру**, **вологість**, **тиск**, **яскравість**.

- Дані використовуються для побудови таблиць, графіків та аналітики у веб-інтерфейсі.

5. Веб-інтерфейс (Frontend)

- Реалізований (той, що на backend частині) за допомогою **Flask + HTML** .
- Дозволяє користувачам:
    - Переглядати всі зібрані дані у вигляді **таблиці**.
    - Отримувати зведену інформацію про параметри середовища.
- Реалізований (той, що на ESP32) за допомогою HTML/CSS/JavaSctript
- Дозволяє користувачам:
    - Переглядати дані в режимі **реального часу**.
- Доступний локально через веб-браузер.

Процеси, що виконує Backend (Python + Flask + MQTT):

1. Отримання та обробка MQTT-повідомлень

- Серверний Python-скрипт підписується на топік MQTT.
- При надходженні повідомлення:
    - JSON або текстові дані розпаршуються.
    - Зберігаються у відповідні поля таблиці бази даних.

2. Зберігання даних у SQLite

- Дані записуються з таймштампом у таблицю.
- Передбачена можливість зчитування всіх записів для виводу на сайт.

3. Обробка HTTP-запитів веб-інтерфейсу

- Flask-роути (/, /data) віддають HTML-сторінки.

- Дані передаються у шаблон та виводяться у вигляді таблиці.

Код back-end частини з поясненнями:

```
function fetchData() {
  fetch('/data')
    .then(response => response.json())
    .then(data => {
      document.getElementById("temp").textContent = data.temp;
      document.getElementById("hum").textContent = data.hum;
      document.getElementById("pres").textContent = data.pres;
      document.getElementById("lux").textContent = data.lux;
    });
}
```

Функція fetchData() надсилає запит до ESP (шлях /data) і отримує актуальні значення температури, вологості, тиску та освітленості. Потім вставляє ці значення у відповідні елементи HTML.

```
function setBME() {
  const delay = document.getElementById("bme").value;
  fetch('/set_bme?val=' + delay);
}
```

Змінює частоту зчитування даних з датчика BME280 на ESP, відправляючи нове значення параметра BME_delay.

Аналогічно для фото та MQTT:

```
function setPhoto() {
  const delay = document.getElementById("photo").value;
  fetch('/set_photo?val=' + delay);
}

function setMQTT() {
  const delay = document.getElementById("mqtt_delay").value;
  fetch('/set_send_mqtt?val=' + delay);
}



document.addEventListener("DOMContentLoaded", function () {
  fetch('/config')
    .then(response => response.json())
    .then(data => {
      document.getElementById("bme").value = data.bme_delay;
      document.getElementById("photo").value = data.photoresistor_delay;
```

```
        document.getElementById("mqtt_delay").value = data.send_mqtt_delay;
      });
  });
```

При завантаженні сторінки одразу запитуються актуальні значення таймерів з ESP (через /config) і підставляються у відповідні поля вводу.

```
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(200, "text/html", index_html); });
```

Обробляє запит на головну сторінку (/) та відправляє HTML-файл.

```
  server.on("/data", HTTP_GET, [](AsyncWebServerRequest *request)
            {
  String json = "{";
  json += "\"temp\":" + String(temperature, 1) + ",";
  json += "\"hum\":" + String(humidity, 1) + ",";
  json += "\"pres\":" + String(pressure, 1) + ",";
  json += "\"lux\":" + String(lux);
  json += "}";
  request->send(200, "application/json", json); });
```

Повертає поточні сенсорні дані у форматі JSON на запит /data.

```
server.on("/config", HTTP_GET, [](AsyncWebServerRequest *request)
            {
  String json = "{";
  json += "\"bme_delay\":" + String(BME_delay) + ",";
  json += "\"photoresistor_delay\":" + String(photoresistor_delay) + ",";
  json += "\"send_mqtt_delay\":" + String(send_mqtt_delay);
  json += "}";
  request->send(200, "application/json", json); });
```

Відправляє клієнту поточні налаштування затримок (delays) для BME280, фоторезистора та відправки через MQTT

```
  server.on("/set_bme", HTTP_GET, [](AsyncWebServerRequest *request)
            {
  if (request->hasParam("val")) {
    BME_delay = request->getParam("val")->value().toInt();
  }
  request->send(200, "text/plain", "OK"); });
```

Оновлює затримку для BME280, якщо параметр val переданий у запиті.

Аналогічно для фото-датчика і MQTT:

```
  server.on("/set_photo", HTTP_GET, [](AsyncWebServerRequest *request)
```

```
          {
  if (request->hasParam("val")) {
    photoresistor_delay = request->getParam("val")->value().toInt();
  }
  request->send(200, "text/plain", "OK"); });

  server.on("/set_send_mqtt", HTTP_GET, [](AsyncWebServerRequest *request)
          {
  if (request->hasParam("val")) {
    send_mqtt_delay = request->getParam("val")->value().toInt();
  }
  request->send(200, "text/plain", "OK"); });
```

```
if (now - photoresistor_tick >= photoresistor_delay)
  {
    adc_raw = analogRead(34);
    float vout = (adc_raw / 4095.0f) * 3.3f;
    if (vout <= 0.0f)
      vout = 0.001f;
    float r_photo = R_FIXED * (3.3f / vout - 1.0f);
    float ratio = R_LUX10 / r_photo;
    float temp = (log10f(ratio) / GAMMA) + 1.0f;
    lux = (uint32_t)powf(10.0f, temp);
    if (lux < 0)
      lux = 0;
    photoresistor_tick = now;
  }
```

Зчитує освітленість за допомогою фоторезистора, перетворюючи аналогове значення з АЦП в люкси:

- analogRead(34) — зчитування сигналу з фоторезистора.
- Розрахунок напруги vout.
- Розрахунок опору фоторезистора r_photo.
- Далі обчислюється значення освітленості (lux) на основі логарифмічної залежності.

```
temperature = data.get('temperature', None)
brightness = data.get('lux', None)
humidity = data.get('humidity', None)
pressure = data.get('pressure', None)
time_ = datetime.now(timezone(timedelta(hours=3))).strftime("%Y-%m-%d %H:%M:%S")
```

Отримання значень сенсорних даних зі словника data, який, ймовірно, надійшов у форматі JSON з повідомлення MQTT.

Формує поточну дату й час зі зміщенням UTC+3, у форматі рядка. Це буде мітка часу (timestamp) для запису в базу даних.

```
c.execute("SELECT COUNT(*) FROM sensor_data")
count = c.fetchone()[0]

max_record = 500
if count >= max_record:
    c.execute("DELETE FROM sensor_data WHERE id = (SELECT id FROM sensor_data
ORDER BY id ASC LIMIT 1)")
```

Підрахунок кількості записів у таблиці sensor_data.

Контроль за розміром таблиці: якщо вже є 500 записів, видаляється найстаріший (той, у кого найменший id), щоб звільнити місце.

```
c.execute('''INSERT INTO sensor_data (timestamp, temperature, brightness,
humidity, pressure)
             VALUES (?, ?, ?, ?, ?)''',
          (time_, temperature, brightness, humidity, pressure))
conn.commit()
```

Вставляє новий запис у таблицю sensor_data з поточним часом та сенсорними значеннями. Після вставки зміни фіксуються через commit().

```
BROKER = "broker.hivemq.com"
PORT = 1883
TOPIC = "esp/sensor/data"
client = mqtt.Client()
client.on_message = on_message
client.connect(BROKER, PORT)
client.subscribe(TOPIC)
client.loop_start()
```

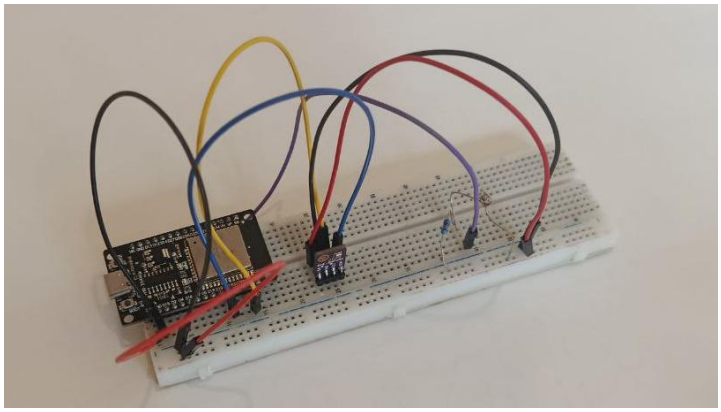
Створення MQTT-клієнта.

Прив’язка обробника on_message — ця функція буде викликатись при надходженні нових повідомлень.

Підключення до брокера та підписка на тему esp/sensor/data.

loop_start() запускає фоновий цикл, який обробляє повідомлення у окремому потоці.

РОЗДІЛ 3. РЕЗУЛЬТАТИ РОБОТИ ПРОЕКТУ



```
Connecting to WiFi..........
Connected to WiFi
192.168.0.159
Attempting MQTT connection...connected
1
data published
Attempting MQTT connection...connected
1
data published
```

Рис. 7. Складений пристрій на макетній платі та дані які він відправляє в консоль

Спочатку пристрій пробує підключитись до WiFi і коли це вийшло то впристрій починає надсилати дані по MQTT на топік який заданий.

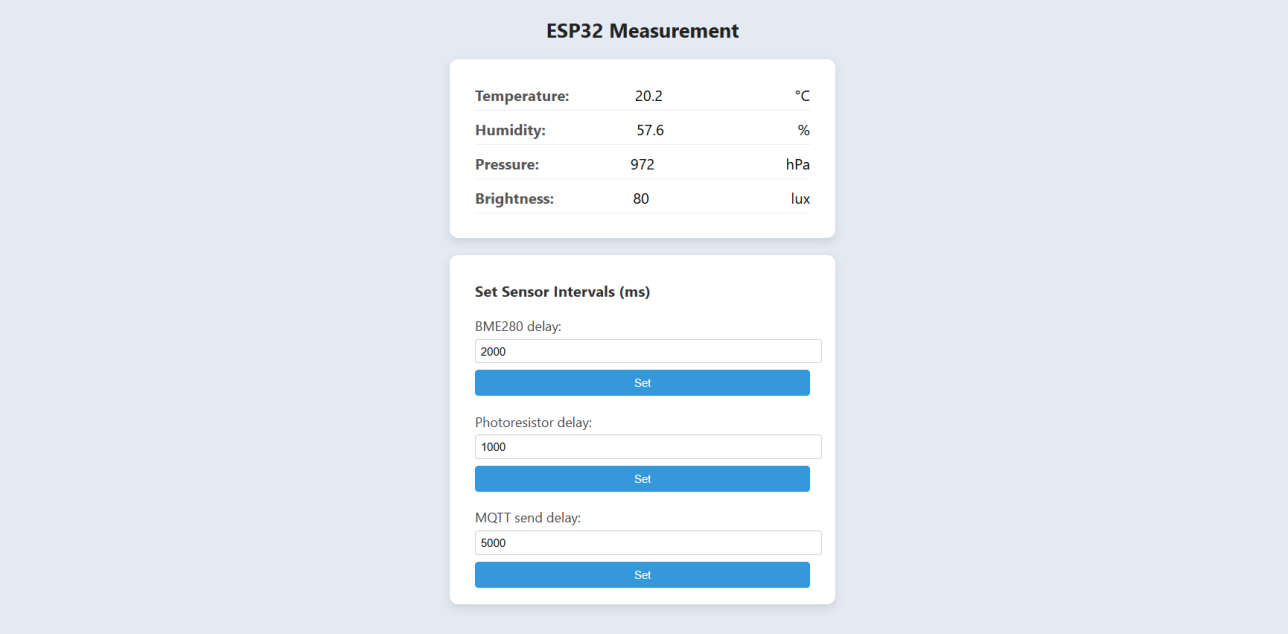
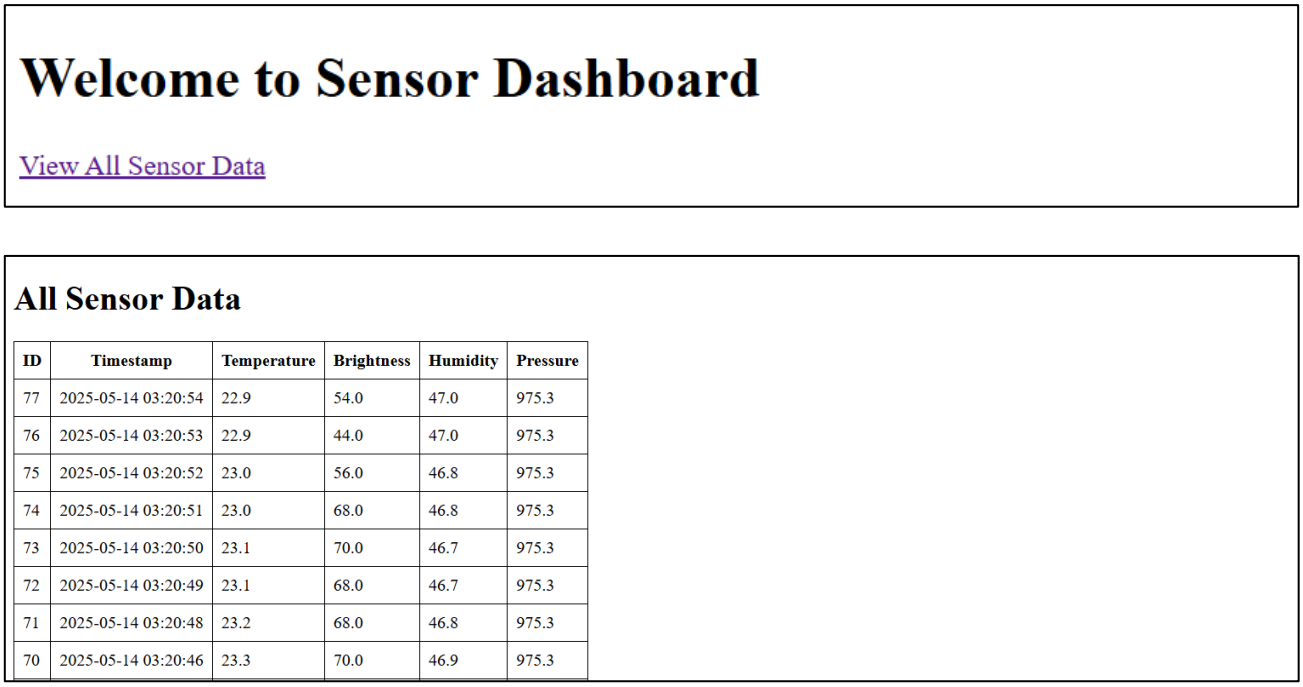


Рис. 8. Зображення роботи веб-сайту на ESP32

На сайті відображаються актуальні дані про рівень вологості, температури, тиску, та рівня освітлення, також на сайті є поля вводу де можна побачити актуальні періоди вимірювання вимірювання температури, вимірювання значення з фоторезистора та надсилання даних по MQTT.



Welcome to Sensor Dashboard

View All Sensor Data

All Sensor Data

ID	Timestamp	Temperature	Brightness	Humidity	Pressure
77	2025-05-14 03:20:54	22.9	54.0	47.0	975.3
76	2025-05-14 03:20:53	22.9	44.0	47.0	975.3
75	2025-05-14 03:20:52	23.0	56.0	46.8	975.3
74	2025-05-14 03:20:51	23.0	68.0	46.8	975.3
73	2025-05-14 03:20:50	23.1	70.0	46.7	975.3
72	2025-05-14 03:20:49	23.1	68.0	46.7	975.3
71	2025-05-14 03:20:48	23.2	68.0	46.8	975.3
70	2025-05-14 03:20:46	23.3	70.0	46.9	975.3

Рис. 9. Візуалізація бази даних на сайті

На цьому сайті відображаються дані що зчитуються з бази даних, сам сайт розміщується локально на комп’ютері, на якому запускається скрипт.

ВИСНОВКИ

Висновки до розділу 1

Основним елементом виступає мікроконтролер ESP32-WROOM-32D, який має двоядерний процесор, підтримку Wi-Fi, що дозволяє реалізувати бездротову передачу даних. Розробка програмної частини здійснюється у середовищі PlatformIO на базі VS Code, що забезпечує гнучкість у розробці прошивки.

Для збору кліматичних даних використовується датчик BME280, який дозволяє точно вимірювати температуру, вологість і атмосферний тиск. Додатково у проєкті використовується фоторезистор, що дозволяє оцінити рівень освітленості навколишнього середовища. Отримані дані передаються через протокол MQTT на комп’ютер, де обробляються серверною частиною, реалізованою на мові програмування Python.

Серверна частина також відповідає за зберігання отриманих даних у базі даних SQLite, що забезпечує можливість подальшого аналізу. Для локального відображення інформації створено вебінтерфейс за допомогою Flask, який надає користувачу зручний доступ до моніторингових даних через браузер.

Висновки до розділу 2

Система складається з мікроконтролера ESP32-WROOM-32D, датчика BME280, фоторезистора, локального веб-інтерфейсу та серверної частини, розробленої на Python з використанням Flask. Мікроконтролер здійснює вимірювання температури, вологості, атмосферного тиску та рівня освітленості, а отримані дані передаються через протокол MQTT на комп’ютер.

На комп’ютері працює сервер, що приймає ці дані та зберігає їх у базі даних SQLite, забезпечуючи можливість подальшого аналізу. Одночасно, веб-

інтерфейс, реалізований за допомогою Flask, дозволяє користувачам переглядати поточні сенсорні дані через браузер.

Інтерфейс також забезпечує можливість зміни параметрів роботи системи, таких як затримка між вимірюваннями для кожного сенсора та частота публікацій MQTT, що підвищує гнучкість і зручність використання пристрою. Веб-сторінка дозволяє у реальному часі отримувати актуальні значення без необхідності перезавантаження чи втручання у код.

У процесі розробки було проведено аналіз ключових компонентів: ESP32 як продуктивного мікроконтролера з підтримкою Wi-Fi, BME280 як багатофункціонального сенсора, та фоторезистора для оцінки рівня освітленості. Було розроблено ефективну структуру бази даних, яка дозволяє зберігати до 500 останніх записів, автоматично видаляючи найстаріші.

Цей проєкт демонструє ефективне використання технологій Інтернету речей (IoT) для автоматизації збору, зберігання та відображення кліматичних даних, що може знайти легко практичне застосування в системах розумного дому, навчальних лабораторіях чи промислових моніторингових рішеннях.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. ESP32 with BME280: https://randomnerdtutorials.com/esp32-bme280-arduino-ide-pressure-temperature-humidity/
2. ESP32 MQTT: https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
3. ESP32 Web Server: https://randomnerdtutorials.com/esp32-web-server-arduino-ide/
4. Photoresistor with ESP32: https://www.keyestudio.com/blog/how-to-use-photoresistor-with-esp32-233
5. GitHub код цієї курсової роботи: