НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»
спеціальності 126 «Інформаційні системи та технології»
на тему: «IoT система для моніторингу стану навколишнього середовища в реальному часі»

Виконав (-ла):
студент IV курсу, групи ІК-91
Бойко Іван Олександрович _____

Керівник:
Професор кафедри інформаційних
систем та технологій,
доктор технічних наук, професор,
Жураковський Богдан Юрійович _____

Рецензент:
Завідувач кафедри цифрового розвитку
Дежавного університету телекомунікацій,
доктор технічних наук, доцент
Жебка Вікторія Вікторівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Бойку Івану Олександровичу

1. Тема проєкту «IoT система для моніторингу стану навколишнього середовища в реальному часі», керівник проєкту Жураковський Богдан Юрійович, доктор технічних наук, професор, затверджені наказом по університету від 31 травня 2023 р. № 2101-с.

2. Термін подання студентом проєкту: 12 червня 2023 року.

3. Вихідні дані до проєкту: наукова література, технічна документація, інсуючі аналоги систем моніторингу стану навколишнього середовища.

4. Зміст пояснювальної записки: огляд предметної області та аналіз аналогічних систем, проєктування системи, розробка програмного забезпечення.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): діаграма послідовностей, структурна схема системи, ER-діаграма БД, алгоритм роботи системи сповіщень.

6. Дата видачі завдання: 27 лютого 2023 року.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження тематики роботи	27.02.2023	
2	Аналіз проблематики	28.02.2023 – 28.03.2023	
3	Дослідження предметної області	29.03.2023 – 10.04.2023	
4	Аналіз аналогічних систем	11.04.2023 – 13.04.2023	
5	Пошук та порівняння технологій для створення системи	14.04.2023 – 17.04.2023	
6	Проєктування системи	17.04.2023 – 30.04.2023	
7	Розробка програмного забезпечення	01.05.2023 – 31.05.2023	
8	Оформлення документації	01.06.2023 – 12.06.2023	

Студент Іван БОЙКО

Керівник Богдан ЖУРАКОВСЬКИЙ

АНОТАЦІЯ

Бойко І.О. IoT система для моніторингу стану навколишнього середовища в реальному часі. КПІ ім. Ігоря Сікорського, Київ, 2023.

Проєкт містить 72 с. тексту, 28 рисунків, 1 таблицю, посилання на 21 літературне джерело та 4 конструкторських документи.

Ключові слова: IoT, навколишнє середовище, моніторинг, комп'ютерна мережа, сервер, база даних.

Об'єктом розробки є IoT система для моніторингу стану навколишнього середовища в реальному часі.

Мета розробки – створення надійної та корисної системи моніторингу стану навколишнього середовища, яка сприяє збереженню навколишнього середовища, забезпечує безпеку та покращує якість життя користувачів.

У дипломному проєкті розроблено базову IoT систему, яка поєднує апаратне забезпечення у вигляді мікроконтролеру та датчику та програмне забезпечення, яке включає в себе серверний і клієнтський додатки та скрипт для роботи апаратного забезпечення. Архітектура системи була спроєктована, опираючись на результати аналізу дослідження предметної області та наявних систем для моніторингу стану навколишнього середовища.

Отримані результати мають потенціал для того, щоб бути корисними при розробці систем охорони навколишнього середовища, розумних міст, промислових центрів тощо.

SUMMARY

Boiko I.O. IoT System for Real-Time Monitoring of the Environment State. Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 2023.

The project consists of 72 pages of text, 28 figures, 1 table, references to 21 literary sources, and 4 design documents.

Keywords: IoT, environment, monitoring, computer network, server, database.

The object of development is an IoT system for real-time monitoring of the environment state.

The aim of the project is to create a reliable and useful system for monitoring the environment state, which contributes to environmental conservation, ensures safety, and improves the quality of life for users. In the diploma project, a basic IoT system has been developed, which combines hardware in the form of a microcontroller and a sensor, and software, including server and client applications, and a script for the operation of the hardware. The architecture of the system was designed based on the results of the analysis of the research in the subject area and existing systems for monitoring the environment state.

The obtained results have the potential to be useful in the development of environmental protection systems, smart cities, industrial centers, etc.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІК91.080БАК.003 ПЗ	Пояснювальна записка	72		
6	А3	ІК91.080БАК.003 Д1	IoT система для моніторингу	1		
7			стану навколишнього			
8			середовища в реальному часі.			
9			Діаграма послідовностей			
10	А3	ІК91.080БАК.003 Д2	IoT система для моніторингу	1		
11			стану навколишнього			
12			середовища в реальному часі.			
13			Структурна схема системи			
14	А3	ІК91.080БАК.003 Д3	IoT система для моніторингу	1		
15			стану навколишнього			
16			середовища в реальному часі.			
17			ER-діаграма БД			
18	А3	ІК91.080БАК.003 Д4	IoT система для моніторингу	1		
19			стану навколишнього			
20			середовища в реальному часі.			
21			Алгоритм роботи системи			
22			сповіщень			
23						
24						
25						
26						
27						
28						

					ІК91.080БАК.003 ТП		
Зм.	Аркуш	№ докум.	Підпис	Дата			
Розроб.		Бойко І.О.			IoT система для моніторингу стану навколишнього середовища в реальному часі. Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Жураковський Б.Ю.				Т	
						КПІ ім. Ігоря Сікорського Група ІК-91	
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «IoT система для моніторингу стану навколишнього середовища в реальному часі»

Київ – 2023 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 4

ВСТУП 5

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ 7

1.1 Загальний огляд об’єкта дослідження 7

1.2 Аналіз та порівняння існуючих систем 10

1.2.1 Aclima Sensing Network 10

1.2.2 IBM Green Horizons 11

1.2.3 Arable Mark 2 12

1.2.4 Libelium Smart Water 12

1.2.5 Cisco Kinetic for Cities 13

1.2.6 Порівняльна характеристика існуючих систем 13

1.3 Постановка задачі 16

Висновки до розділу 18

2 ПРОЄКТУВАННЯ СИСТЕМИ 19

2.1 Загальна схема взаємодії компонентів системи 19

2.2 Апаратне забезпечення системи 23

2.2.1 Вибір технології зв’язку 24

2.2.2 Вибір апаратного забезпечення системи 26

2.2.3 ESP32 26

2.2.4 DHT22 датчик 27

2.2.5 Аналіз обраного апаратного забезпечення 28

2.2.6 Середовище проєктування апартного забезпечення системи 29

2.2.7 Проєктування апартного забезпечення системи 32

2.3 Архітектура програмного забезпечення 33
2.3.1 Визначення та задачі архітектури програмного забезпечення 33
2.3.2 Принципи створення програмного забезпечення SOLID 36
2.3.3 DDD (Domain-driven design) 39
2.3.4 Гексагональна архітектура 42
2.3.5 Поєднання DDD підходу та гексагональної архітектури 44
2.4 Вимоги до інтерфейсу користувача 45
Висновки до розділу 46
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 48
3.1 Опис стеку технологій 48
3.1.1 Мова програмування TypeScript 48
3.1.2 Платформа NodeJs 50
3.1.3 СУБД PostgreSQL 52
3.1.4 React 54
3.2 Скриптування апаратного забезпечення додатку 56
3.3 Розробка серверної частини додатку 61
3.4 Розробка бази даних 62
3.5 Розробка клієнтської частини додатку 64
Висновки до розділу 68
ВИСНОВКИ 69
ПЕРЕЛІК ПОСИЛАНЬ 71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІМ (PWM, або Pulse Width Modulation) – це техніка, яка дозволяє контролювати потужність, подану на електронні пристрої, шляхом зміни ширини імпульсів в електричному сигналі.

API (Application Programming Interface) – програмний інтерфейс додатку.

ООП – об'єктно-орієнтоване програмування.

DX (Developer Experience) – досвід розробки.

ОС – операційна система.

СУБД - система управління базами даних.

DOM (Document Object Model) – јб'єктна модель документа.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертекстових документів.

JSON (JavaScript Object Notation) – це текстовий формат обміну даними між комп’ютерами.

HTML (HyperText Markup Language) – мова розмітки гіпертексту.

URL (Uniform Resource Locator) – адреса ресурсу.

ВСТУП

Сьогодні настав час, коли технологічний розвиток є надзвичайно швидким і масштаб його розповсюдження є настільки стрімким, що важко знайти сферу життя сучасної людини, на яку б він значно не вплинув. Такому процесу технологічного розвитку значно посприяла четверта промислова революція, ключовими характеристиками якої є:

– Взаємодія – активність, яка здійснюється різними системами спільно.

– Віртуалізація – створення штучних об'єктів та середовищ.

– Децентралізація – планування та виконання процесів незалежно від центральної частини системи, або її повна відсутність.

– Режим реального часу – виконання процесів у режимі реального часу.

– Орієнтація на поточне обслуговування – під час створення системи, думка кожного окремого споживача впливає на кінцевий продукт.

– Модульність – створення системи із різних незалежних модулів, що дає можливість швидко та зручно їх замінювати, не вплинувши на інші частини системи.

Інтернет речей (IoT) – основна складова технологія четвертої промислової революції, яка відповідає більшості з її характеристик, значно розширює та спрощує зв’язок людей із технологіями та усім, що їх оточує.

Саме мережа інтернет дає можливість існувати технологіям, що базуються на IoT, які використовує інтернет для об’єднання незалежних складових системи, вільного обміну даними між ними тощо. IoT системи здатні як спрощувати та оптимізувати повсякденні завдання людей, так і налаштовувати та створювати масштабні процеси виробництва, моніторингу та аналізу даних.

Особливу увагу в контексті IoT заслуговують системи моніторингу стану навколишнього середовища в реальному часі. Такі системи відіграють важливу роль в контролі та прогнозуванні стану навколишнього середовища, попередження небезпечних ситуацій та реагування на них. Вони також допомагають в оптимізації ресурсів, економії енергії та підвищенні продуктивності. Завдяки IoT, ми можемо

отримувати деталізовані дані про стан середовища, що підвищує нашу здатність впливати на його якість та стабільність.

З огляду на глобальні виклики, такі як зміни клімату, забруднення навколишнього середовища та вичерпання природних ресурсів, тема моніторингу стану навколишнього середовища набуває особливого значення. Здатність відстежувати та аналізувати ці процеси в реальному часі стає критичною для нашого виживання та сталого розвитку. Для нашої держави, основним викликом є повоєнна відбудова України як сучасної та розвиненої європейської держави. Перед нашим поколінням стоять задачі реконструкції та зведення нових міст, цивільної та стратегічної інфраструктури тощо. Для майбутнього нашої країни надзвичайно важливо проводити розбудову за новітніми тенденціями IoT, використовуючі такі технології, як, наприклад, Smart City. В такому контексті сучасності, IoT системи, що дозволяють збирати, обробляти та аналізувати дані про стан навколишнього середовища, відіграють важливу роль.

Метою цієї роботи є створення системи, що забезпечить збір, обробку та аналіз даних про навколишнє середовище, що дозволить своєчасно реагувати на зміни та вживати необхідних заходів.

В рамках цієї роботи, планується розробити концепцію системи, визначити технічні та програмні засоби для її реалізації, проаналізувати потенційні виклики та перешкоди на шляху її впровадження.

Ця дипломна робота має на меті не лише проєктування IoT системи для моніторингу стану навколишнього середовища, але й розуміння ширшого контексту впровадження таких систем в нашому суспільстві.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний огляд об’єкта дослідження

Моніторинг навколишнього середовища є ключовим компонентом багатьох галузей, включаючи екологію, сільське господарство, метеорологію та багато інших. Моніторинг допомагає виявляти шкідливі зміни в навколишньому середовищі, що дозволяє вчасно реагувати на них.

До основних завдань моніторингу навколишнього середовища належать:

– Відслідковування змін навколишнього середовища

– Моделювання та прогнозування антропогенного впливу на навколишнє середовище

– Об’єктивне оцінювання стану навколишнього середовища

– Передбачення змін у навколишньому середовищі.

При цьому можна виділити основні принципи, на яких будуються, та яким мають відповідати системи моніторингу навколишнього середовища:

– Об'єктивність та достовірність – будь-яка система, що збирає та обробляє дані, має бути протестована та стабільна. При аналізі даних мають враховуватися похибки та інші впливи.

– Систематичність – так як збір даних, їх подальша обробка та аналіз також є частинами процесу моніторингу стану навколишнього середовища, то важливо, щоб система діяла систематично. В такому випадку, можна отримати значно об’єктивнішу та повнішу інформацію про стан середовища.

– Узгодженність технічного та програмного забезпечення – для роботи системи важливо правильно налаштувати взаємодію її складових. Наприклад, сенсори системи моніторингу передають інформацію до ядра системи, використовуючи певний контракт для цієї взаємодії.

– Комплексність оцінки – оцінка стану має враховувати усі особливості та деталі об’єкту моніторингу, сторонні впливи, похибки тощо.

Моніторинг стану навколишнього середовища можна поділити на два основних види: дистанційний та наземний [1].

Дистанційний моніторинг стану навколишнього середовища полягає в використанні супутникових зображень, аерофотозйомки, радарів, та інших подібних технологій для віддаленого збору даних про середовище. Цей метод включає обробку цих даних за допомогою спеціалізованого програмного забезпечення для отримання інформації про стан навколишнього середовища на великих просторах або у важкодоступних місцях. Дистанційний моніторинг особливо корисний для вивчення змін клімату, дефорестації, забруднення води, моніторингу природних катастроф та ін.

Наземний моніторинг включає безпосередній збір даних на місці, за допомогою використання таких інструментів, як метеостанції, буєві, зонди, проби води, зразки грунту тощо. Він може бути дуже точним і детальним, дозволяючи отримувати дані про конкретні місця та параметри навколишнього середовища. Наземний моніторинг може бути використаний для збору даних про якість повітря, якість води, стан ґрунту, біорізноманіття та ін.

Важливо зазначити, що ці два типи моніторингу часто використовуються разом для отримання найбільш повної та точної інформації про стан навколишнього середовища.

Структура системи моніторингу змін навколишьного середовища складається із наступних компонентів: «Спостереження», «Оцінка фактичного стану», «Прогноз стану довкілля», «Оцінка прогнозованого стану» та «Підтримка прийняття управлінських рішень» [2]. Ці компоненти зв’язані між собою прямими та зворотніми зв’язками (рис. 1.1).

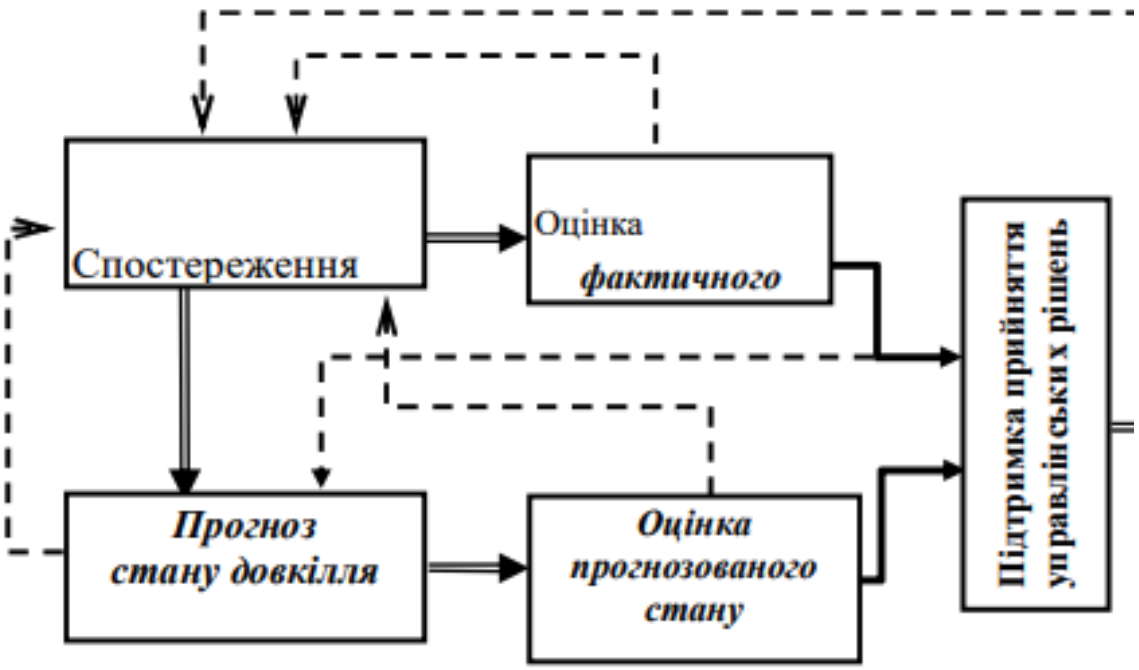

Рисунок 1.1 – Складові системи моніторингу довкілля

Впровадження IoT в цей процес має багато переваг, наприклад:

– Більш точний та ефективний збір даних: IoT сенсори можуть працювати безперервно, збираючи дані з високою точністю та здатні надавати дані в реальному часі. Вони можуть вимірювати широкий спектр показників, включаючи температуру, вологість, рівень CO2, рівень шуму, якість води та багато іншого.

– Автоматизація: IoT технології можуть автоматизувати процес збору даних, що зменшує необхідність вручну виконувати цей процес. Це знижує витрати на персонал та зменшує можливість помилок.

– Моніторинг у реальному часі: IoT дозволяє отримувати дані в реальному часі, що дозволяє оперативно реагувати на небезпечні зміни в навколишньому середовищі.

– Великі дані та аналітика: IoT сенсори генерують велику кількість даних, які можна аналізувати для отримання глибоких та цінних висновків про стан навколишнього середовища. Це може допомогти в прийнятті кращих рішень щодо охорони та управління навколишнім середовищем.

– Прогнозування та попередження: Із використанням аналітики великих даних та штучного інтелекту, дані, отримані від IoT сенсорів, можуть бути використані для прогнозування та попередження небезпечних ситуацій.

– Забезпечення прозорості та просування освіти: Дані з IoT-сенсорів можуть бути доступні громадськості, що допомагає забезпечити прозорість в діях уряду та бізнесу щодо охорони довкілля. Вони також можуть служити важливим інструментом для освіти та підвищення обізнаності про екологічні питання.

Незважаючи на численні переваги, впровадження IoT технологій у сферу моніторингу стану навколишнього середовища також має деякі недоліки, про які варто пам’ятати та намагатися мінімізувати їх:

– Витрати на впровадження – встановлення та обслуговування IoT-сенсорів може бути вартісним. Обслуговування може включати в себе витрати на купівлю обладнання та його установку, підтримку та заміну тощо. Також є досить значними витрати на зберігання та обробку великих обсягів даних та розробку програмног забезпечення.

– Цілісність даних – IoT сенсори можуть бути ламатися, давати непередбачені похибки, виходити із ладу від навколишніх впливів. Це може призвести до втрати даних або неправильного відображення стану навколишнього середовища.

– Кібербезпека – IoT сенсори та системи передачі даних можуть бути вразливі до хакерських атак або несанкціонованого доступу. Це може призвести до порушення приватності, втрати або зміни даних.

– Залежність від енергії – IoT сенсори потребують джерел енергії для роботи. У віддалених чи важкодоступних місцях це може становити проблему.

1.2 Аналіз та порівняння існуючих систем

1.2.1 Aclima Sensing Network

Aclima Sensing Network – це інноваційна IoT система, розроблена для моніторингу якості повітря. Ця система використовується для збору даних про якість повітря в різних місцях, включаючи міські, промислові та сільські

місцевості. Aclima використовує високоточні датчики, які збирають дані про концентрацію різних забруднювачів, включаючи вуглецевий діоксид, оксиди азоту, озон, пил і т.д.

Датчики, що використовуються в системі Aclima, розміщуються в стратегічних місцях для забезпечення максимального покриття. Вони використовують передові бездротові технології для передачі зібраних даних на центральні сервери Aclima, де вони обробляються і аналізуються. Оброблені дані потім доступні через веб-інтерфейс або мобільний додаток, що дозволяє користувачам відстежувати якість повітря в реальному часі.

Окрім забезпечення надійного моніторингу якості повітря, Aclima також використовує технології машинного навчання для виявлення шаблонів та трендів в даних. Ця інформація може слугувати важливим ресурсом для урядових органів та організацій, що працюють над покращенням якості повітря і зменшенням забруднення довкілля. За допомогою цих даних, вони можуть розробляти та впроваджувати стратегії, що допоможуть в боротьбі зі змінами клімату та забрудненням повітря.

1.2.2 IBM Green Horizons

IBM Green Horizons – це великий і амбіційний проект, що використовує передові технології для прогнозування та управління якістю повітря. Система базується на IoT датчиках, що збирають дані про якість повітря в різних точках міста. Зібрані дані потім передаються на сервери IBM, де вони обробляються за допомогою алгоритмів машинного навчання.

Відмінною особливістю IBM Green Horizons є її можливість прогнозувати майбутню якість повітря, використовуючи історичні дані та поточні тенденції. Ця особливість робить IBM Green Horizons цінним інструментом для урядових органів та організацій, що працюють над питаннями сталого розвитку і контролю забруднення повітря.

За допомогою цієї системи, вони можуть отримувати точні прогнози про майбутній стан якості повітря та розробляти ефективні стратегії для боротьби зі забрудненням. Все це веде до збільшення ефективності використання ресурсів і зменшення впливу на довкілля.

1.2.3 Arable Mark 2

Arable Mark 2 – це високотехнологічний IoT датчик, який використовується для моніторингу стану сільськогосподарських культур і навколишнього середовища. Він вимірює різні параметри, включаючи температуру, вологість, освітленість, тиск, вітер, і навіть активність дощу і фотосинтезу.

Датчики Arable Mark 2 вбудовані в міцний корпус, що захищає їх від негативного впливу навколишнього середовища. Вони використовують бездротову технологію для передачі зібраних даних на центральний сервер для обробки та аналізу.

Однією з ключових особливостей Arable Mark 2 є його здатність адаптуватися до різних сільськогосподарських умов. Він може вимірювати різні параметри, які впливають на здоров'я та продуктивність рослин, включаючи температуру, вологість, освітленість, тиск та інше. Всі ці дані допомагають фермерам краще розуміти умови росту своїх культур та планувати свою діяльність.

1.2.4 Libelium Smart Water

Libelium Smart Water – це IoT-рішення, розроблене для моніторингу якості води. Система складається з водонепроникних датчиків, які можна встановити в будь-якому водоймі, та центрального сервера для обробки та аналізу даних.

Датчики Libelium Smart Water вимірюють різні параметри, включаючи рівень розчиненого кисню, провідність, pH, температуру та інші показники, що відображають якість води. Зібрані дані потім передаються через бездротову мережу на сервер для аналізу.

Libelium Smart Water використовує передові технології для забезпечення точного та надійного моніторингу якості води. Завдяки цьому, організації, що відповідають за управління водоймами, можуть отримувати точні дані про якість води в реальному часі та вживати необхідні заходи для забезпечення безпечності і здоров'я людей.

1.2.5 Cisco Kinetic for Cities

Cisco Kinetic for Cities – це рішення IoT, яке надає комплексне рішення для моніторингу різних аспектів міського середовища. Ця система забезпечує збір, обробку та аналіз даних з різних джерел, включаючи датчики якості повітря, датчики шуму, метеорологічні станції та інше.

Система Cisco Kinetic for Cities базується на модульній архітектурі, що дозволяє легко додавати нові датчики і розширювати функціональність системи. Кожний датчик передає дані через бездротову мережу на центральний сервер, де вони обробляються та аналізуються. Така архітектура робить систему гнучкою та простою для розширення.

Основна перевага системи Cisco Kinetic for Cities полягає в тому, що вона надає повні і точні дані про стан міського середовища в реальному часі. Це дозволяє міським управлінням та іншим організаціям швидко реагувати на зміни умов та приймати вчасні рішення.

Ця система також використовує алгоритми машинного навчання для виявлення шаблонів та трендів у даних. Наприклад, вона може виявити певні шаблони забруднення повітря або зміни в рівнях шуму. Ця інформація може бути використана для вдосконалення планування міста і керування ресурсами.

1.2.6 Порівняльна характеристика існуючих систем

Далі (табл. 1.1) наведено порівняльну характеристику існуючих систем моніторингу стану навколишнього середовища за деякими базовими параметрами.

Таблиця 1.1 – Порівняльна характеристика існуючих систем.

Назва	Вимірювані параметри	Тип зв’язку	Архітектура	Інтеграція з іншими системами
Aclima Sensing Network	Вуглецевий діоксид, оксиди азоту, озон, пил та інше	Wi-Fi / LTE / ZigBee	Централізована, з датчиками, що передають дані на сервер	Вбудований API для інтеграції з іншими системами
IBM Green Horizons	Різні параметри якості повітря	Wi-Fi / LTE	Облачна архітектура з можливістю розширення	Підтримка IoT рішень IBM
Arable Mark 2	Температура, вологість, освітленість, тиск, вітер, активність дощу, фотосинтез	LoRaWAN, LTE	Централізована, з датчиками, що передають дані на сервер	Вбудований API для інтеграції з іншими системами
Libelium Smart Water	Рівень розчиненого кисню, провідність, pH, температура та інші параметри якості води	ZigBee, LoRaWAN, Wi-Fi, 3G/4G	Модульна архітектура з бездротовими датчиками	Вбудований API для інтеграції з іншими системами

Cisco’s Kinetic for Cities	Якість повітря, шум, метеорологічні дані та інше	Wi-Fi / LTE	Централізована, з датчиками, що передають дані на сервер	Вбудований API для інтеграції з іншими системами

Порівнюючи характеристики цих систем IoT для моніторингу стану навколишнього середовища, можна зробити наступні висновки:

– Aclima Sensing Network відрізняється високоточними датчиками і використанням машинного навчання для виявлення шаблонів забруднення повітря. Ця система підходить для широкого спектра місць і потребує високої вартості встановлення та обслуговування. Має високу свідомість безпеки даних і використовує широкий набір заходів для забезпечення безпеки. Це включає використання шифрування для захисту передачі даних, аутентифікацію користувачів та контроль доступу до системи. Додатково, система має заходи для захисту фізичної безпеки датчиків та серверів.

– IBM Green Horizons використовує передові технології машинного навчання для прогнозування якості повітря в майбутньому. Ця система має потужні аналітичні можливості і облачну архітектуру, що дозволяє розширювати її функціональність. Покладає великий акцент на безпеку даних. Вона використовує захищені протоколи комунікації та шифрування даних, що передаються по мережі. Додатково, система має механізми для аутентифікації та авторизації користувачів, що дозволяє контролювати доступ до системи та її функціональності.

– Arable Mark 2 зосереджена на моніторингу сільськогосподарського середовища, здатна вимірювати різні параметри, які впливають на рослини. Ця система має високу адаптивність і може працювати в різних умовах, але її вартість може бути середньою до високої. Враховує безпеку даних, забезпечуючи шифрування даних та безпечний обмін даними через захищені мережеві протоколи. Вона також використовує механізми автентифікації та авторизації, щоб забезпечити захист від несанкціонованого доступу до системи та зібраних даних.

– Libelium Smart Water спеціалізується на моніторингу якості води і пропонує надійну систему для точного вимірювання параметрів води в різних водоймах. Ця система підтримує різні типи бездротового зв’язку і має можливості інтеграції з іншими системами. Використовує шифрування та захищені протоколи передачі даних для забезпечення конфіденційності та цілісності даних. Система також надає можливості аутентифікації та авторизації, що дозволяє контролювати доступ до системи та забезпечити безпеку даних.

– Cisco’s Kinetic for Cities забезпечує повнофункціональну систему моніторингу міського середовища з використанням алгоритмів машинного навчання для виявлення шаблонів та трендів. Ця система має високу вартість встановлення та обслуговування, але вона надає високу якість даних та широкі можливості інтеграції. Має вбудовані механізми безпеки, такі як шифрування, аутентифікація та контроль доступу. Вона використовує різні рівні захисту, включаючи захист від кібератак та зловживань. Додатково, система надає інструменти для моніторингу та аналізу безпеки, що дозволяє виявляти та відповідати на потенційні загрози безпеці даних.

1.3 Постановка задачі

Після аналіз предметної області об’єкту дослідження доцільно створити основні функціональні та нефункціональні вимоги до системи. Це необхідо для перевірки та аналізу результату дослідження, створення висновків тощо.

Система має відповідати наступним функціональним вимогам:

– Збір даних – система повинна забезпечувати збір даних з різних сенсорів, включаючи такі параметри, як температура, вологість тощо.

– Обробка даних – система повинна вміти обробляти великі обсяги даних, що надходять від сенсорів, і приводити їх до формату, який придатний для аналізу та візуалізації.

– Візуалізація даних – система повинна забезпечувати візуалізацію даних про стан навколишнього середовища в реальному часі, що допомагає користувачам легко інтерпретувати ці дані.

– Алерти та сповіщення – система повинна мати здатність встановлювати певні порогові значення для різних параметрів та відправляти сповіщення користувачам, коли ці значення перевищуються.

Система має відповідати наступним нефункціональним вимогам:

– Надійність – система повинна бути надійною та забезпечувати постійний збір даних незалежно від зовнішніх умов.

– Масштабованість – система повинна бути гнучкою та масштабованою, здатною підтримувати додавання нових сенсорів або розширення функціоналу.

– Безпека – система повинна забезпечувати безпеку зберігання та передачі даних, включаючи захист від несанкціонованого доступу, змін або втрати даних.

– Швидкість роботи – система повинна бути в змозі обробляти великі обсяги даних від сенсорів в реальному часі без затримок або зниження продуктивності.

– Ефективне використання ресурсів – система повинна ефективно використовувати ресурси, включаючи мережеві ресурси, ресурси зберігання даних та обчислювальні ресурси.

– Зручність використання – інтерфейс системи повинен бути інтуїтивно зрозумілим та легким у використанні для користувачів з різним рівнем технічної грамотності.

Аби створити IoT систему для моніторингу стану навколишнього середовища в реальному часі, необхідно:

– Спроєктувати архітектуру системи – розробити оптимальну архітектуру IoT системи, обрати оптимальні сенсори, способи та методи передачі даних. Створити архітектуру програмного забезпечення, обрати оптимальні технології.

– Розробити алгоритм обробки та передачі даних – розробити алгоритми для обробки даних з сенсорів, їх передачі та візуалізації в реальному часі.

– Розробити серверну частину системи – використовуючи обрані технології та алгоритми, розробити серверну частину системи.

– Розробити серверну частину системи – розробити інтерфейс користувача, який зручним та зрозумілим для користувача чином відобразить дані, що зберігає та оброблює система.

– Протестувати систему – провести комплексне тестування системи, включаючи тестування надійності, коректності відображення даних, а також зручності використання системи користувачами.

Висновки до розділу

В цьому розділі було проведено роботу щодо вивчення предметної області та загальних принципів роботи систем моніторингу стану навколишнього середовища. Визначені основні характеристики, види та цілі моніторингу стану навколишнього середовища. Під час роботи над розділом були описані основні недоліки та переваги інтеграції IoT технологій у системи моніторингу стану навколишнього середовища. Визначено, що інтеграція IoT систем в обрану сферу є доцільною, перспективною та ефективною.

Для глибшого розуміння теми були описані та проаналізовані основні характеристики та особливості найпопулярніших існуючих систем. Проведено порівняльну характеристику цих систем на основі декількох обраних параметрів. Під час дослідження цих систем, ціна для їх встановлення та підтримки розглядалася, як важлива характеристика. Проте, вирішено не додавати цю характеристику в порівняльну таблицю, тому що ціна може сильно варіюватися в залежності від конкретної конфігурації системи, кількості датчиків, потреб зберігання даних, вимог до пропускної здатності тощо. Крім того, ціни часто можуть бути договірними і залежати від обсягу покупки та термінів контракту.

За результатами дослідження створено функціональні та нефункціональні вимоги до системи. Створена та описана послідовність етапів створення системи.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Загальна схема взаємодії компонентів системи

Усі складові частини системи можна поділити на функціональні групи за їх основними характеристиками, ролями та призначенням:

– Апаратне забезпечення – множина фізичних компонентів системи.

– Серверне програмне забезпечення – ядро системи, яке зберігає та обробляє дані користувача, зберігає та виконує бізнес логіку додатку тощо.

– Клієнтське програмне забезпечення – програмне забезпечення, що відповідає за роботу інтерфейсу користувача. Взаємодія користувача та системи відбувається саме через клієнтську частину програмного забезпечення.

Зобразимо схему взаємодії цих функціональних груп в рамках роботи системи моніторингу стану навколишнього середовища в реальному часі на рис. 2.1.

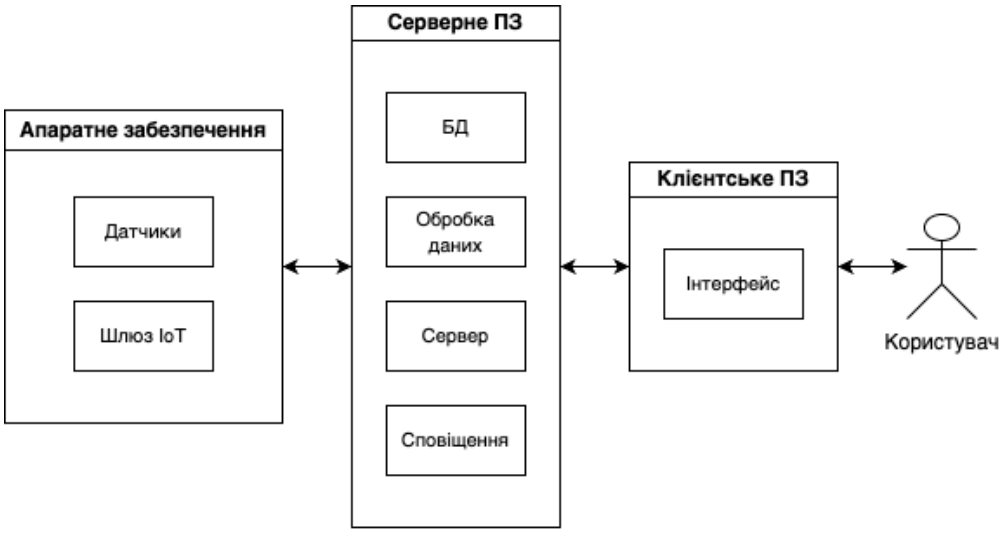

Рисунок 2.1 – Функціональні групи системи

На основі вивченого теоретичного матеріалу було створено загальну схему компонентів системи (рис. 2.2). Система має складатися із наступних компонентів:

– Датчики – це вузли, які збирають дані про зовнішнє середовище. Вони включають датчики температури, вологості, тиску та освітленості. Кожен з цих датчиків збирає відповідні дані і передає їх до шлюзу IoT.

– Шлюз IoT – це пристрій, який отримує дані від датчиків і передає їх до хмари.

– Хмара – це місце, де зберігаються і обробляються дані. Хмара аналізує дані на сервері обробки даних і надсилає результати на користувацький інтерфейс. Хмара також керує датчиками, шлюзом IoT та системою повідомлень.

– Сервер обробки даних – це сервер, який аналізує дані, отримані від шлюзу IoT через хмару.

– Користувацький інтерфейс – це інтерфейс, через який користувач може переглядати результати аналізу даних.

– Користувач – це особа, яка використовує систему. Користувач може переглядати результати, керувати системою через хмару і переглядати історію даних.

– Система повідомлень – це система, яка отримує повідомлення від датчиків через хмару і надсилає їх користувачу.

– База даних – це місце, де зберігаються дані. Хмара керує зберіганням даних в базі даних.

– Історія даних – це записи про минулі дані, які зберігаються в базі даних. Користувач може переглядати історію даних.

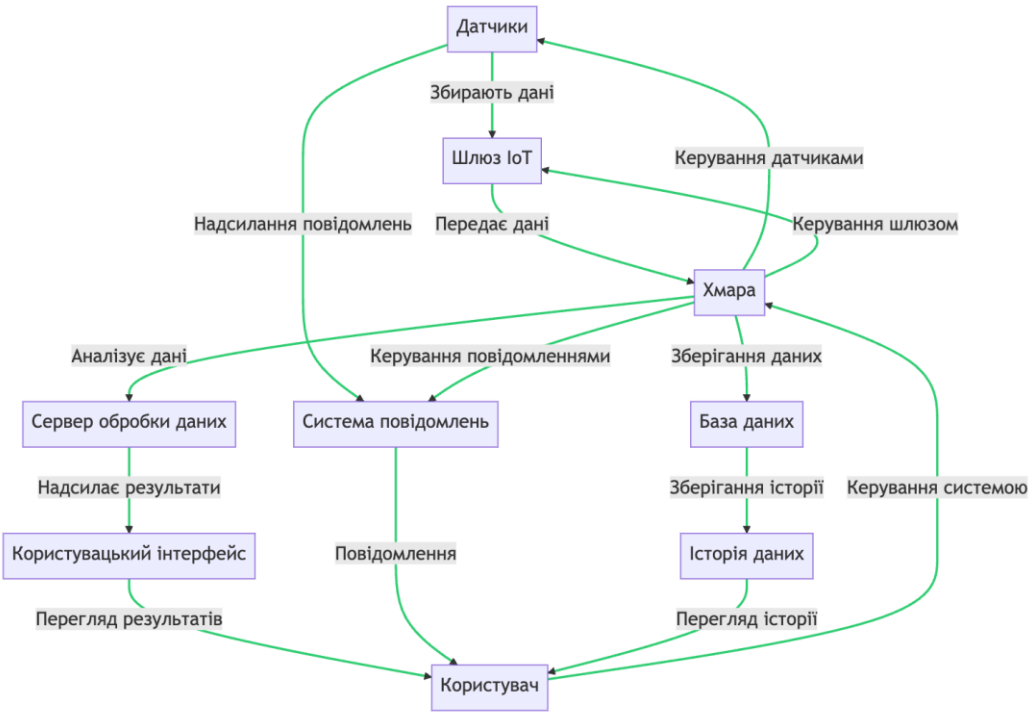

Рисунок 2.2 – Складові системи моніторингу

Зв'язки між компонентами можна описати наступним чином:

– Датчики і Шлюз IoT – датчики збирають дані про зовнішнє середовище і передають ці дані до шлюзу IoT. Шлюз IoT служить як мост між датчиками і хмарою.

– Шлюз IoT і Хмара – шлюз IoT передає дані, отримані від датчиків, до хмари для подальшого аналізу і зберігання.

– Хмара і Сервер обробки даних – хмара аналізує дані на сервері обробки даних і надсилає результати на користувацький інтерфейс.

– Сервер обробки даних і Користувацький інтерфейс – сервер обробки даних аналізує дані і надсилає результати на користувацький інтерфейс для перегляду користувачем.

– Користувацький інтерфейс і Користувач – користувацький інтерфейс дозволяє користувачу переглядати результати аналізу даних.

– Користувач і Хмара – користувач може керувати системою через хмару, включаючи керування датчиками, шлюзом IoT, системою повідомлень і базою даних.

– Хмара і Система повідомлень – хмара керує системою повідомлень, яка отримує повідомлення від датчиків і надсилає їх користувачу.

– Хмара і База даних – хмара керує зберіганням даних в базі даних.

– База даних і Історія даних – база даних зберігає історію даних, яку користувач може переглядати.

На основі зв’язків між компонентами системи можна створити діаграму станів системи. На рис. 2.3 зображні можливі стани роботи системи та переходи між ними.. Описуючи діаграму станів, ми можемо виділити наступні елементи:

– Бездіяльність – це початковий стан системи, коли вона ще не розпочала збирати дані або обробляти їх. У цьому стані система очікує активації датчиків або початку збору даних.

– Збір – цей стан відображає активний процес збору даних з датчиків. Система отримує вхідні дані з датчиків про температуру, вологість та якість повітря.

– Обробка – після збору даних система переходить до стану обробки даних. В цьому стані дані аналізуються і обробляються для отримання корисних висновків або виявлення аномалій.

– Сповіщення – кщо під час обробки даних виявляються аномалії або критичні зміни, система переходить до стану сповіщення. В цьому стані система надсилає сповіщення користувачеві про виявлену проблему або зміну стану навколишнього середовища.

– Бездіяльність – після відправки сповіщення система повертається до стану бездіяльності і очікує наступного циклу збору та обробки даних.

Рисунок 2.3 – Діаграма станів

2.2 Апаратне забезпечення системи

Апаратне забезпечення є основою будь-якої IoT системи. Апаратне забезпечення IoT системи включає датчики, пристрої IoT, модулі зв'язку, а також системи живлення та інфраструктуру. Загалом, ці компоненти працюють разом для збору, передачі, обробки та відображення даних про навколишнє середовище [3].

Датчики є ключовою складовою системи моніторингу. Вони перетворюють фізичні параметри, такі як температура, вологість, рівень шуму, якість повітря та освітленість, на вимірювані дані, які потім можна обробити та аналізувати. Важливо вибрати сенсори, які забезпечують точні та надійні вимірювання, а також сумісні з обраними пристроями шлюзу IoT.

Пристрої IoT, такі як Arduino Uno, Arduino Mega, Raspberry Pi, ESP8266 або ESP32, служать мозком системи. Вони зчитують дані з сенсорів, обробляють їх, а потім передають на сервер або в хмару [4]. Вибір відповідного пристрою IoT залежить від потреб та характеристик системи. На вибір можуть впливати такі характеристики, як: обчислювальна потужність, можливості зв'язку, підтримка сенсорів та бюджет.

Модулі зв'язку, такі як Wi-Fi, Bluetooth, LoRa, ZigBee, забезпечують зв'язок між пристроями IoT та сервером або хмарою. Вибір технології зв'язку залежить від вимог до дальності, швидкості передачі даних, споживання енергії та сумісності з іншими компонентами системи.

Системи живлення забезпечують необхідну енергію для роботи системи. Це може бути живлення від мережі, батареї або навіть відновлювані джерела, такі як сонячні панелі. Це особливо важливо для пристроїв IoT, які розміщені у віддалених місцях або у складних для доступу областях.

Інфраструктура, включає в себе всі необхідні механічні або електронні компоненти для монтажу, захисту та підключення сенсорів та пристроїв IoT. Це можуть бути корпуси, кріплення, кабелі та інше.

2.2.1 Вибір технології зв’язку

Зважаючи на різноманітність IoT систем і проєктів, є досить багато поширених способів зв’язку, які використовують апаратні компоненти систем.

Мережі, через які з’єднуються IoT системи бувають:

– Бездротові мережі – мережі, які використовують радіочастотні зв'язки для передачі даних між пристроями. Через відсутність фізичних пристроїв з’єднання вони дають змогу гнучко себе розташовувати, маючи при цьому великий радіус дії. Бездротові мережі є широко використовуваними в IoT системах, зокрема для збирання даних про стан навколишнього середовища. Іншою перевагою є простота роботи та налаштування бездротових мереж, які дають змогу відносно легко підключати багато пристроїв в одну мережу та налаштовувати взаємодію між ними. До недоліків бездротових мереж можна віднести досить обмежену пропускну здатність та вразливість цих мереж до впливу екстремально складних погодних умов. Основними типами бездротових мереж є Wi-Fi, Bluetooth і Zigbee:

1. Wi-Fi є популярним бездротовим стандартом, який використовується для безпроводового з'єднання пристроїв до мережі Інтернет. Використовує радіохвилі для передачі даних на відстань до кількох сотень метрів. Wi-Fi є

простим в налаштуванні, підтримує високу швидкість передачі даних та працює на великій кількості пристроїв.

2. Bluetooth – є бездротовим стандартом, який використовується для локального безпроводового з'єднання пристроїв. Використовується для передачі даних на невеликій відстані.

3. Zigbee є бездротовим стандартом, який підтримується на великій кількості IoT пристроїв. Має досить велику дальність роботи та підтримує одночасне підключення великої кількості пристроїв.

– Дротові мережі – мережі, які використовують фізичні кабелі для з’єднання пристроїв системи. Основними типами дротових мереж є Ethernet і RS-485:

1. Ethernet – підтримує високу швидкість передачі даних і використовується для підключення IoT пристроїв до локальної мережі.

2. RS-485 є стандартом передачі даних за допомогою диференційного сигналу на дротовій мережі. Використовується звичайно для підключення IoT пристроїв у вимогливих промислових середовищах, де великі відстані та шум можуть вплинути на якість сигналу.

До основних протоколів передачі даних належать:

– LPWAN (Low-Power Wide Area Network) є технологією передачі даних, яка спеціально розроблена для IoT пристроїв з низьким споживанням енергії [5]. Вона дозволяє передавати дані на великі відстані з довгим часом роботи без необхідності у зарядці. До її недоліків можна віднести досить низьку швидкість передачі даних та обмежену пропускну здатність цієї технології.

– MQTT (Message Queuing Telemetry Transport) – протокол передачі повідомлень, розробленим для обміну даними між пристроями через мережі з низьким рівнем пропускної здатності або ненадійні з'єднання [6]. MQTT має вбудовані механізми для гарантування доставки повідомлень, включаючи "Quality of Service" (QoS) та збереження сеансів. MQTT широко використовується для передачі даних з датчиків та керування пристроями в розподілених системах IoT. До недоліків цього протоколу можна віднести низьку швидкість доставки повідомлень при великій кількості підключених пристроїв.

Опираючись на результати аналізу способів зв’язку IoT систем, вирішено обрати бездротове з’єднання, а саме Wi-Fi, через його високу мобільність, простоту використання та великий радіус дії – ці параметри є важливими для віддалених систем моніторингу стану навколишнього середовища. Аби забезпечити передачу даних в повному об’ємі, без їх втрат навіть при перебоях з’єднання, вирішено обрати протокол MQTT, який має механізми, що гарантують доставку повідомлень у системі.

2.2.2 Вибір апаратного забезпечення системи

Враховуючи вибір технології для зв’язку, створення IoT системи моніторингу стану навколишнього середовища в реальному часі, базуватиметься на мікроконтролері ESP32, який має вбудований модуль Wi-Fi, що значно спрощує пррєктування апаратної частини системи, адже відкидає необхідність встановлення додаткового модулю Wi-Fi. Іншою важливою первагою є популярність цього сенсору та велика кількість навчальних матеріалів щодо його використання. Для вимірювання температури та вологості системи було обрано сумісний із мікроконтролером датчик DHT22, який є поширеним через свою точність та надійність.

2.2.3 ESP32

ESP32 – це високопродуктивний мікроконтролер, розроблений компанією Espressif Systems [7]. Він був спеціально розроблений для інтелектуальних приладів та Інтернету речей (IoT). Серед основних характеристик та особливостей ESP32 можна виділити:

– Двоядерний процесор – ESP32 має двоядерний процесор Xtensa 32-bit LX6, що працює з частотою до 240 MHz. Кожне ядро може бути незалежно контрольовано або використане сумісно для виконання важких обчислень.

– Wi-Fi та Bluetooth – ESP32 включає повністю інтегрований 802.11 b/g/n Wi-Fi та Bluetooth 4.2 модулі. Це забезпечує високу підключуваність для IoT-пристроїв.

– Пам'ять – ESP32 має внутрішню SRAM, а також дозволяє зовнішнє розширення пам'яті за допомогою QSPI-інтерфейсу. Залежно від моделі, ESP32 може мати вбудовану пам'ять flash різного розміру.

– GPIO – ESP32 має до 36 загальних портів вводу/виводу (GPIO), з можливістю виконання аналогового вводу, аналогового виводу, ШІМ, I2C, SPI та інших протоколів.

– Інтегровані сенсори – багато моделей ESP32 має вбудовані сенсори, такі як сенсор температури, сенсор залиття, а також зчитувач аналогових сигналів (ADC) і модулятор ширини імпульсу (PWM).

– Захист інформації – ESP32 має апаратний криптографічний засіб, що включає апаратну підтримку AES, SHA-2, RSA-4096, Elliptic Curve Cryptography (ECC), випадковий генератор чисел (RNG), а також апаратні ключі захисту від нападів.

Як недолік цього мікроконтролеру можна виділити його споживання енергії – хоча ESP32 має режими енергозбереження, його загальне споживання енергії може бути вищим за деякі інші мікроконтролери, особливо при використанні Wi-Fi та Bluetooth. Це може стати проблемою для пристроїв, що працюють від батарей.

2.2.4 DHT22 датчик

DHT22 – це надійний датчик вологості та температури, що широко використовується в IoT проектах. Цей датчик також відомий як AM2302. DHT22 відмінно підходить для вимірювання температури та вологості в навколишньому середовищі. Він є одним з найточніших сенсорів, доступних на ринку, і має високу стабільність.

Далі наведені основні характеристики та функції DHT22 [8]:

– Температурний діапазон – DHT22 може вимірювати температуру від -40 до +125 градусів за Цельсієм з точністю ±0.5°C.

– Діапазон вологості – датчик може вимірювати вологість від 0 до 100% з точністю ±2-5%.

– Живлення: DHT22 живиться від 3,3 до 6 вольт, що робить його сумісним з більшістю мікроконтролерів, включаючи Arduino та ESP8266.

– Робоча частота – DHT22 вимірює температуру та вологість приблизно кожні 2 секунди, що забезпечує достатню швидкість вимірювань для більшості застосувань.

2.2.5 Аналіз обраного апаратного забезпечення

Переваги обраного апаратного забезпечення:

– Система надає високу точність вимірювання температури та вологості.

– Мобільність – використання сонячної панелі і Li-Po батареї дозволяє системі працювати у віддалених місцях. Система також є відносно компкатною.

– Arduino має велику та активну спільноту розробників, яка надає багато відкритих ресурсів для навчання та підтримки.

– ESP32 і DHT22 – це перевірені часом компоненти, що відрізняються високою надійністю та довговічністю.

– Завдяки використанню ESP32, система має великий простір для кастомізації та модифікації. ESP32 дозволяє легко додавати додаткові сенсори або модулі, що дозволяє адаптувати систему до вимог, які можуть часто змінюватися.

– Компоненти, які використовуються в цій системі, є відносно недорогими. Крім того, з використанням сонячної панелі вартість експлуатації системи в подальшому буде ще нижчою, хоч і варто зауважити, що в такому випадку система потребуватиме більших початкових інвестицій

Недоліки та обмеження обраного апаратного забезпечення:

– Обмеження датчика DHT22 – незважаючи на високу точність, DHT22 має обмеження вимірювання температури та вологості. Це може стати проблемою в екстремальних умовах, де температура або вологість виходить за межі діапазону датчика.

– Для використання цієї системи потрібні навички програмування та електроніки.

– Навіть використання сонячних панелей може не вирішити усіх проблем із живленням та автономністю системи, адже вона може не працювати ефективно в умовах недостатнього освітлення або при низькій температурі.

2.2.6 Середовище проєктування апартного забезпечення системи

Дослідження аналогів системи, та навчальної літератури дало змогу знайти найефективніший та найшвидший спосіб набуття, навичок роботи із ESP32, знань про його інфраструктуру та основні особливості навичок проєктувати ситеми. Таким способом є використання віртуальних інструментів для симуляції різних мікроконтролерів та модулів, включаючи датчики та інші периферійні пристрої, які використовуються при побудові IoT систем. До основних переваг віртуальних систем симуляціїї можна включити:

– Безпека – працюючи з фізичним обладнанням, завжди є ризик замкнення, перегріву або інших проблем, які можуть пошкодити обладнання. З віртуальною симуляцією, кожен має змогу експериментувати та тестувати код без ризику пошкодження обладнання.

– Економія часу та ресурсів – замість купівлі та налаштування фізичного обладнання, віртуальна система дає змогу швидко та безкоштовно конфігурувати та тестувати систему.

Однією із найпопулярніших систем для віртуальної симуляціїї є сервіс Wokwi, до основних переваг якого належать наступні пункти:

– Висока інтерактивність та симуляція в реальному часі – Wokwi дозволяє симулювати роботу проекту в реальному часі, що означає, що користувач може

бачити, як його код впливає на пристрої в реальному часі. Це не лише допомагає зрозуміти, як працює код, але й дозволяє виявити та виправити помилки.

– Велика бібліотека компонентів – Wokwi має велику бібліотеку компонентів, яку можна використовувати в своїх проектах. Це означає, що користувач може експериментувати з різними типами сенсорів, модулів та інших компонентів, не купуючи їх.

– Спільнота та підтримка – Wokwi має активну спільноту користувачів та розробників, які готові допомогти з будь-якими питаннями або проблемами. Це може бути дуже корисним, особливо для новачків.

Сервіс Wokwi складається з декількох основних компонентів, які спільно створюють інтерактивне середовище для моделювання та симуляції проектів на базі різних платформ:

– Інтерактивний редактор коду (рис. 2.4) – це основний інструмент для розробки програмного забезпечення для симуляцій. Редактор підтримує мови програмування C/C++ для Arduino та MicroPython для ESP32 та Raspberry Pi Pico. Він також має функції підсвічування синтаксису, автодоповнення та віртуального логування для спрощення процесу кодування.

– Симулятор апаратного забезпечення (рис. 2.5) – дозволяє користувачу моделювати різноманітне апаратне забезпечення, включаючи мікроконтролери (Arduino Uno, ESP32, ESP8266), дисплеї (LED, LCD, OLED), датчики, модулі тощо. Він показує, як код контролює апаратне забезпечення в реальному часі, що дозволяє користувачу зрозуміти, як працює його проект.

– Бібліотека компонентів (рис. 2.6) – Wokwi має велику бібліотеку доступних для симуляції компонентів. Користувач може обрати компоненти, які потрібні для кожного його проекту, та додати їх до симуляції.

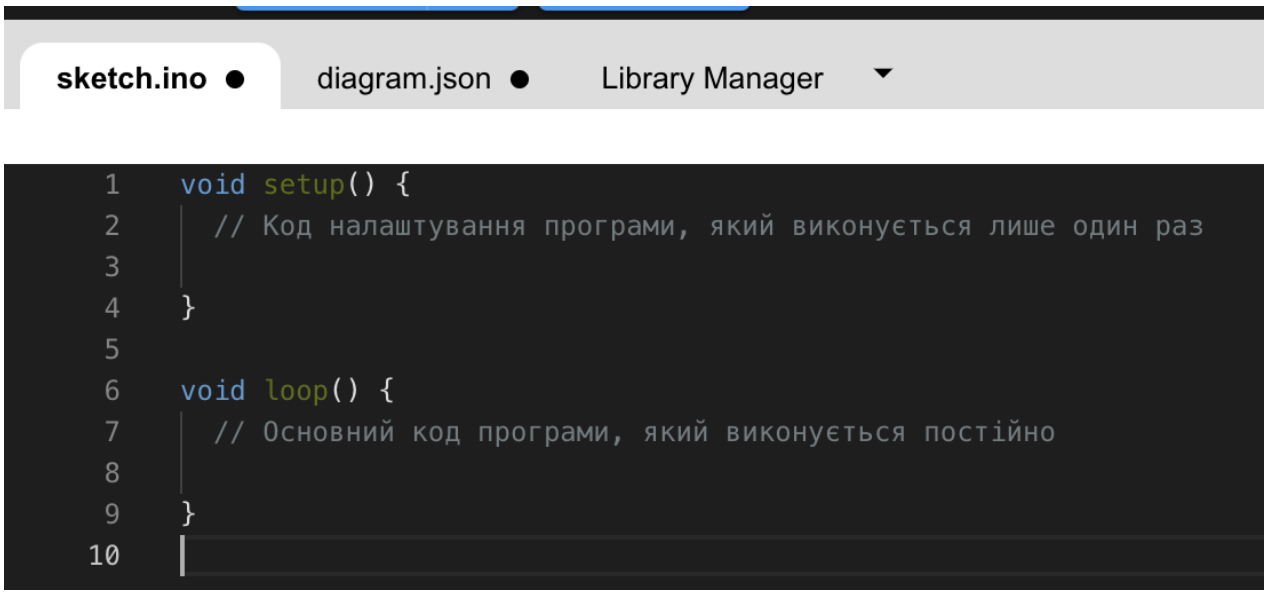

Рисунок 2.4 – Інтерактивний редактор коду сервісу Wokwi

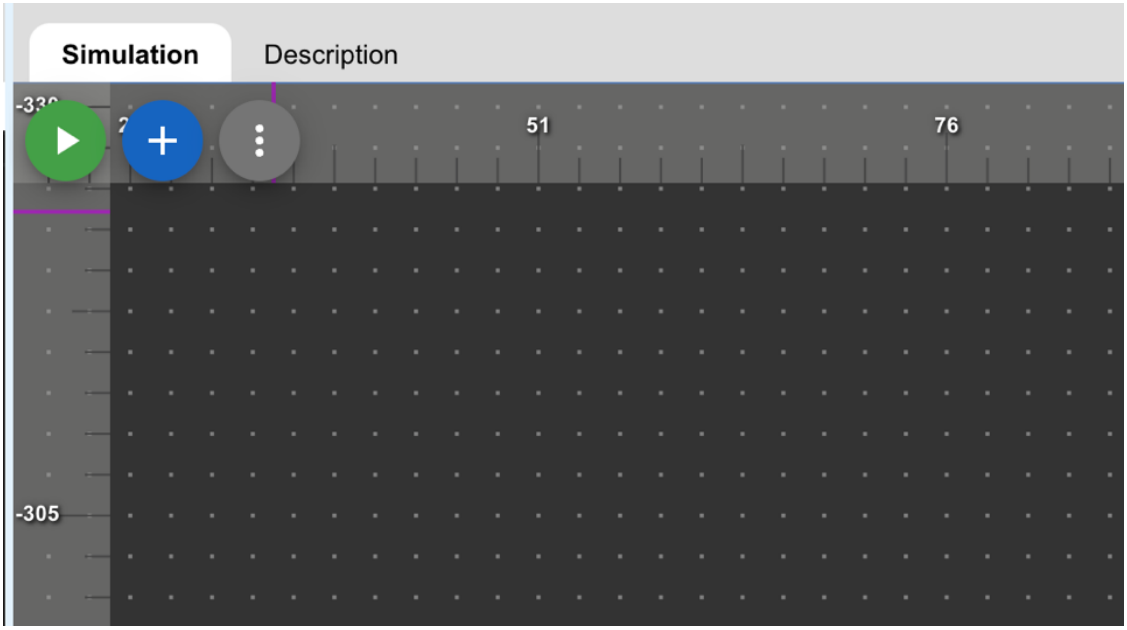

Рисунок 2.5 – Симулятор апаратного забезпечення сервісу Wokwi

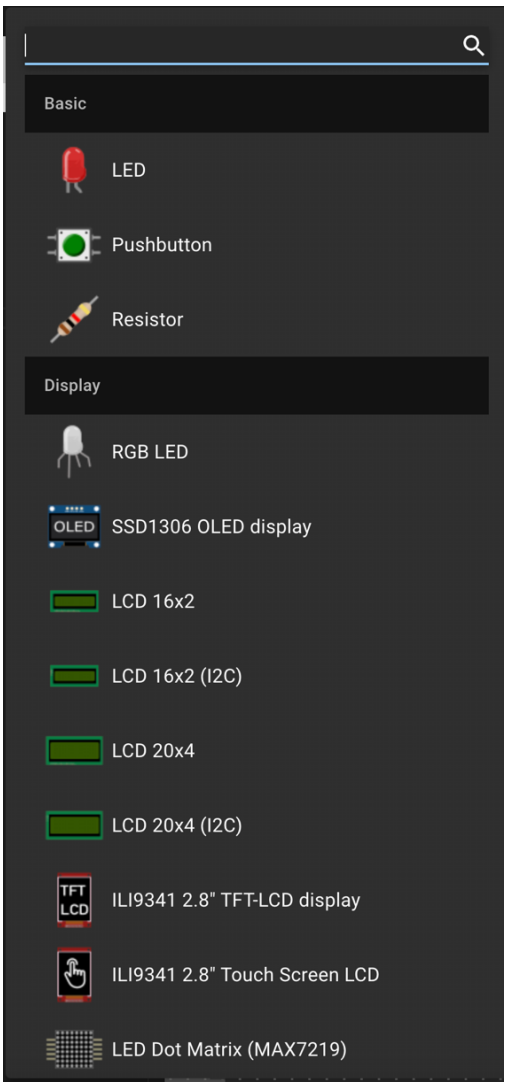

Рисунок 2.6 – Бібліотека апаратного забезпечення сервісу Wokwi

2.2.7 Проєктування апартного забезпечення системи

Спроєктуємо апаратне забезпечення, що складається із вище обраних складових. Першим кроком необхідно обрати компоненти з бібліотеки компонентів Wokwi та додати їх на робочу поверхню симулятора. Далі побудуємо схему, з’єднавши наші компоненти. Побудована схема зображена на (рис. 2.7).

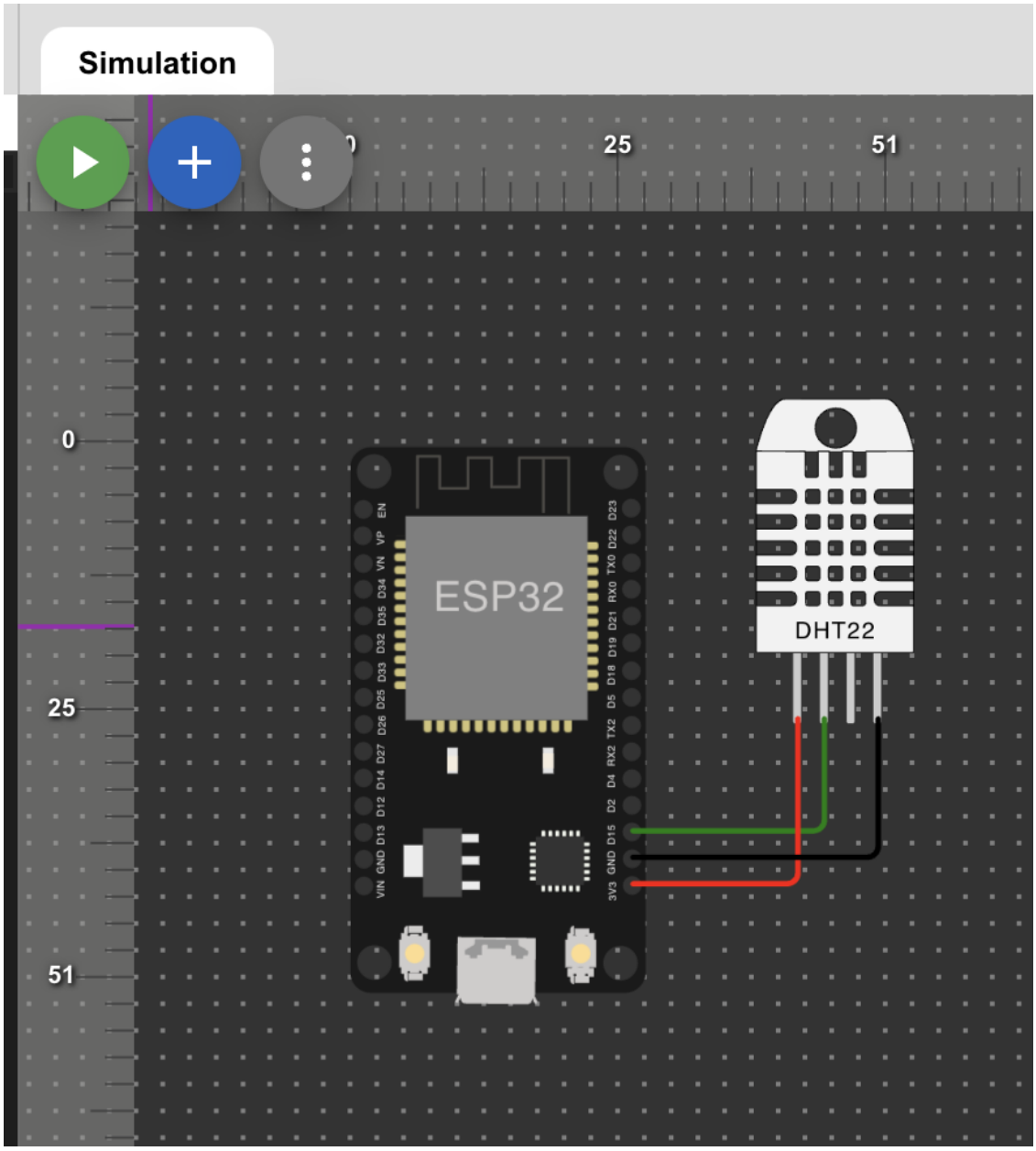

Рисунок 2.7 – Проєкт апаратного забезпечення системи

2.3 Архітектура програмного забезпечення

2.3.1 Визначення та задачі архітектури програмного забезпечення

Архітектура програмного забезпечення є фундаментальним організаційним вираженням системи, яке охоплює структуру, поведінку та більш високий рівень її організації. Вона представляє загальне бачення системи, надаючи концептуальний каркас, за основою якого відбувається розробка системи.

Використання добре пропрацьованої під потреби системи архітектури програмного забезпечення сприяє створенню надійних та ефективних програм, забезпечуючи швидку, надійну та просту взаємодію усіх компонентів системи. Архітектура визначає, як компоненти системи взаємодіють між собою, деталізує обов'язки кожного компоненту та визначає, як різні компоненти системи зберігають та обробляють дані.

Архітектура програмного забезпечення також включає в себе визначення взаємодій з користувачами, іншими системами та сервісами. Вона охоплює технічні рішення, що впливають на працездатність, надійність, масштабованість та безпеку системи.

Ця дисципліна має стратегічне значення, оскільки правильно визначена архітектура може сприяти успіху проекту, зменшуючи ризики, пов'язані з розробкою програмного забезпечення, і забезпечуючи відповідність кінцевого продукту до вимог бізнесу та користувачів. Без ясно визначеної архітектури, команди можуть стикнутися з невизначеністю та неконсистентністю у процесі розробки, що може призвести до збоїв, перевищення бюджету, невдалого виконання вимог або навіть закриття проєктів.

Варто зазначити, що архітектура не є і не може бути статичною. Вона повинна еволюціонувати разом з системою та адаптуватися до змін у бізнес-вимогах та технологіях. Ознакою гарної архітектури є легкість її трансформаціїї під нові потреби бізнесу та проєкту.

Основними характеристиками архітектури програмного забезпечення є такі параметри як:

– Масштабованість – означає здатність системи працювати ефективно при збільшенні навантаження, що може включати кількість користувачів, обсяг даних або кількість транзакцій. Для IoT систем масштабованість є особливо важливою, оскільки кількість пристроїв може швидко зростати. Наприклад, в системі моніторингу навколишнього середовища, датчики можуть розміщуватися у все більшій кількості місць, що збільшує обсяг даних для обробки. Отже, програмне забезпечення має бути здатне ефективно обробляти цей зростаючий обсяг даних.

Це може досягатися за допомогою таких методів, як розподілена обробка даних, використання обчислювальних хмар та використання баз даних великих обсягів. Поняття масштабованість також може вживатися не тільки в контексті збільшення об’ємів оброблюваної інформації, але й у сенсі розширення функціональності самої системи. В цьому контексті, термін означає, що внесення змін та додавання нових бізнес вимог до системи має бути простим, стандартизованим, та безпечним. Безпечність внесення змін означає, що вірогідність некоректної роботи системи при внесені нових змін має бути максимально малою.

– Відмовостійкість – означає здатність системи відновлюватися після збоїв без втрати даних або зниження продуктивності. В IoT системах, які залежать від постійного потоку даних від датчиків, це особливо важливо. Наприклад, якщо відбувається збій у зв'язку з погіршенням або повною зупинкою роботи одного із датчиків, система повинна мати можливість швидко відновити зв'язок та продовжити нормальну роботу. В ідеальному випадку, система взагалі не має припиняти свою роботу, при неполадках окремих її частин. Це може бути досягнуто за допомогою резервного копіювання даних, автоматизованих процедур відновлення тощо.

– Безпека є важливою вимогою до будь-якої системи, що збирає, зберігає або передає відомості. У системах IoT це особливо критично, оскільки дані, що збираються, можуть бути особистими або чутливими. Окрім цього, вразливі місця в безпеці системи можуть бути використані для атак зловмисників на систему, що може призвести до відмови в обслуговуванні або несанкціонованого доступу. Отже, архітектура програмного забезпечення повинна включати заходи безпеки, такі як шифрування, аутентифікація, авторизація тощо.

– Модульність – в архітектурі програмного забезпечення допомагає спрощувати процеси розробки, тестування та обслуговування. Це досягається шляхом розділення програми на незалежні частини або модулі, кожен з яких виконує специфічну функцію. Наприклад, у системі моніторингу навколишнього середовища може бути модуль для збору даних з датчиків, модуль для обробки цих даних та модуль для відображення результатів користувачам. Взаємодія між

модулями повина відбуватися за завчасно визначенимим чіткими контрактами. Це потрібно для того, аби зменшити зв’язність компонентів системи між собою, тобто зробити їх більш незалежними. Такий підхід може використовуватися не лише під час створення програмного забезпечення, а й при проєктуванні будь-яких складних багатокомпонентних систем. Він допомагає спрощувати заміну та внесення змін до окремих частин системи та робити ці процеси більш безпечними, надає змогу виконувати зміни в різних частинах системи паралельно, значно прискорюючи розробку систми. Також модульність значно спрощує розуміння системи, адже за дотримання цієї вимоги, кожна функціональна частина системи є сконцентрованою лише в одному місці, при цьому, деталі реалізаціїї цієї функціональності зовсім не обов’язково ретельно вивчати – контракт, за яким працює будь-який модуль, вже дає змогу зрозуміти, яку саме функціональність виконує модуль, та як її можна використати.

– Ефективність є ключовою вимогою до архітектури програмного забезпечення, яка забезпечує оптимальне використання ресурсів. Це включає в себе ефективне використання обчислювальних ресурсів, пам'яті, мережі та сховища даних. Наприклад, в системі IoT, що обробляє великі обсяги даних від датчиків, ефективна обробка даних може включати в себе використання ефективних алгоритмів, що зменшують навантаження на процесор та пам'ять, а також оптимізацію передачі даних для зменшення використання мережевого трафіку.

2.3.2 Принципи створення програмного забезпечення SOLID

Принципи SOLID – це набір сформованих та ефективних практик написання ефективного та зрозумілого коду. Цей набір практик сформований у вигляді перліку правил, які протягом багатох років використовували та формували на власному досвіді покоління розробників програмного забезпечення [9]. Ці правила були сформовані внаслідок колективного досвіду роботи над проектами в абсолютно різних сферах розробки програмного забезпечення. Роберт С. Мартін, відомий автор книг на тему ефективної розробки програмного забезпечення,

сформулював та назвав ці правила. Метою цих принципів є уникнення найбільш поширених проблем, з якими зустрічаються розробники програмного забезпечення:

– Внесення змін до кодової бази породжує ще більшу кількість змін, які не пов’язані із новою функціональністю.

– Нові зміни до кодової бази ламають стару функціональність.

Програмне забезпечення, що відповідає принципам SOLID, легше та дешевше підтримувати, легше зрозуміти, швидше розробляти в команді та легше тестувати. Дотримання цих принципів не гарантує успіху, але уникнення їх у більшості випадків призведе до принаймні неоптимальних результатів з точки зору функціональності, вартості та якості.

SOLID включає в себе наступні принципи:

– Принцип єдиного обов’язку – цей принцип закликає концентрувати частини програмного забезпечення за їх функціональністю. Наприклад, в класі існувати лише методи, які виконують спільне завдання, а інші методи необхідно перемістити за його межі. Цей принцип допомагає чітко розділяти код за зонами відповідальності, та скоріше в ньому орієнтуватися. Якщо програмне забезпечення створене із дотриманням цього принципу, то для змінення певної функціональності програмісту необхідно модифікувати код лише в одному місці, а не поширювати свої зміни по всій кодовій базі.

– Принцип відкритості/закритості – цей принцип стверджує, що модулі програмного забезпечення необхідно проєктувати таким чином, щоб вони були відкриті для розширення шляхом додавання нового коду, а не модифікації вже існуючого. Основним маркером програмного забезпечення, що написане без дотримання прицнипу відкритості/закритості, є випадки, коли для додавання нової функціональності необхідно розширяти декілька подібних конструкцій умовних операторів (if-else, switch тощо) у різних частинах модуля. Зазвичай, конструкції умовних операторів, які необхідно розширити, мають великий розмір та є складними для розуміння. А якщо, не додати нову умову хоча б в один з них – ПЗ спрацює некоректно. Принцип відкритості/закритості закликає додавати нову

функціональність не вглиб існуючого модуля, а вшир, що дозволяє ізолювати зміни та зробити їх більш безпечними. Система, яка створена із дотриманням цього принципу, є гнучкою та простішою для розуміння, вартість її підтримки та розвитку значно зменшується.

– Принцип підстановки Лісков – цей принцип стосується проєктування програмного забезпечення за допомогою парадигми ООП. Принцип стверджує, що при проєктуванні класів необхідно виконувати наслідування таким чином, що об’єкти, які наслідуються від основного класу, не мають порушувати його поведінку. Основним маркером порушення даного принципу є перевизначення класами-потомками методів базового класу – часто це означає, що наслідування у цьому випадку варто уникнути, так як логіка базового класу не підходить для вирішення конкретної задачі. В цьому прикладі, також частково порушується прицип відкритості/закритості, адже замість додавання нової логіки, змінюється вже її існуюча частина. Цей принцип також пов’язаний із принципом розділення інтерфейсу, який буде описаний далі. Для дотримання цього принципу достатньо слідкувати за використанням наслідування та навчитися визначати випадки, в яких його варто уникнути, замінивши введеням нових інтерфейсів, класів об’єктів тощо.

– Принцип розділення інтерфейсу – цей принцип тісно пов’язаний із попереднім принципом, прото є більш загальним, охоплює питання більш високорівневого проєктування системи. Він закликає створювати інтерфейси таким чином, щоб об’єкти, які їх реалізовували, використовували їх повністю. Характерною ознакою порушення цього принципу є проєктування системи за допомогою малої кількості великих інтерфейсів. При цьому, сутності, що створюються за допомогою цих інтерфейсів, не використовують усі поля та методи від них. Це призводить до переускладнення системи та зростання кількості помилок, адже об’єкти, які мають забезпечувати лише малі частини функціональності системи, наслідуються від інтерфейсів, що змушують їх виконувати усю функціональність. В таких випадках порушується і прицип єдиного обов’язку.

– Принцип інверсії залежностей – цей принцип також стосується взаємодії модулів. Цей принцип закликає реалізовувати взаємодію модулів таким чином, щоб вони залежали не один від одного, а від абстракцій, тобто умовних контрактів взаємодії між ними. При цьому, контракти взаємодії не мають залежати від внутрішніх реалізацій модулів – при змінні модулів, контракти мають залишатися чинними. Це дозволяє спроєктувати систему таким чином, що зміна реалізації одного модуля ніяк не вплине на інші модулі, які його використовують, адже ці модулі використовують змінений модуль за чітко визначеним контрактом, їх не хвилює, як він організований всередині. При розробці системи із дотриманням принципу інверсії залежностей, спочатку визначається структура модулів та контракти їх взаємодії, а вже потім ці модулі програмно реалізуються. Дотримання цього принципу дозволяє створити надійну та гнучку систему, яку відносно легко адаптувати до змін вимог та додавання нової функціональності.

Принципи SOLID найчастіше пояснюються та використовуються за допомогою ООП-мов із статичною типізацією (Java, C# тощо). Також більшість прикладів у літературі описують використання цих принципів для створення серверних додатків. Незважаючи на це, принципи SOLID є універсальними та загальними, вони не залежать від напряму розробки, мови програмування чи набору технологій та здатні адаптуватися у будь-яку сферу розробки програмного забезпечення та мати на неї значний вплив.

2.3.3 DDD (Domain-driven design)

DDD (Domain-drive design) – це підхід до розробки ПЗ, який пропонує структуру організації бізнес логіки та інших компонентів системи за шарами [10]. На рис 2.8 зображено структуру організації слоїв у DDD підході.

Ядром системи є доменний шар. В контексті DDD шар домену включає в себе бізнес логіку та сутності, до яких вона відноситься. Доменний шар включає наступні ключові елементи:

– Сутності – об'єкти, що мають стійкий ідентичний стан протягом часу. Наприклад, користувач системи може бути представлений як сутність.

– Об'єкти значення – незмінні об'єкти, які використовуються для представлення простих елементів домену. Вони не мають свого власного ідентифікатора, і їх рівність визначається на основі їх значень. Наприклад, адреса може бути представлена як об'єкт значення.

– Агрегати – група об'єктів (сутностей та об'єктів значення), які розглядаються як єдине ціле. Вони забезпечують гарантії щодо консистентності бізнес-правил.

– Сервіси домену – операції, що не належать жодній конкретній сутності або агрегату. Вони містять важливу бізнес-логіку, яка не відображається натурально в доменних об'єктах.

– Події домену – повідомлення про важливі події, що відбулися в домені, і які можуть впливати на стан домену.

Шар додатку займається поєднанням та оркестрацією бізнес логіки із доменного шару та сутностей із інфраструктурного шару. Роботу шару додатку можна порівняти з відомим патерном програмування фасад, який інкапсулює в собі деяку логіку із взаємодії багатьох сутностей та дає спрощений інтерфейс для використання.

Інфраструктурний шар містить код, який допомагає різним частинам системи спілкуватися одна з одною, таким як код для взаємодії з БД, файловою системою, мережею тощо. Цей шар також може містити адаптери, які перетворюють дані з формату, який використовується в доменному шарі, у формат, який може використовувати зовнішнє середовище (наприклад, БД). В багатьох реалізаціях в склад інфраструктурного шару включають шар репозиторіїв. Репозиторій – це абстракція, яка відповідає за отримання, збереження чи пошуку доменних сутностей. Репозиторії дозволяють відділити бізнес логіку від логіки роботи з данними, наприклад роботу з БД.

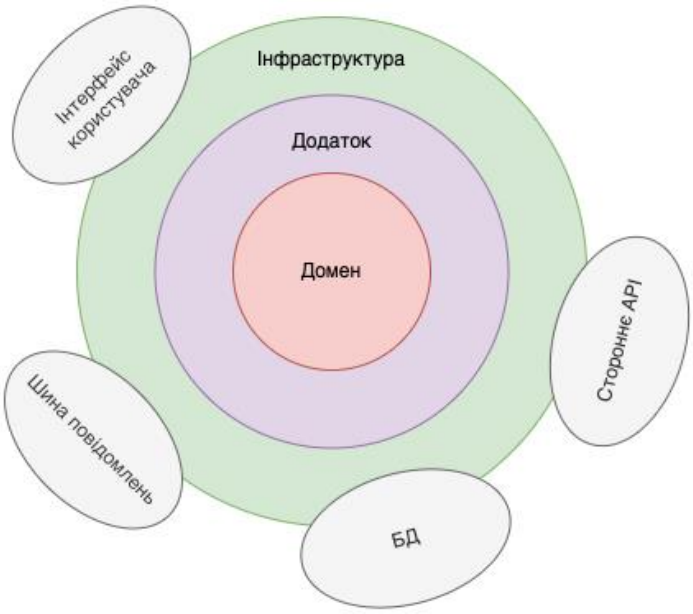

Рисунок 2.8 – Структура DDD системи

До переваг DDD належать:

– Фокус на бізнес логіку – DDD дозволяє командам розробників зосереджуватися на бізнес-логіці та бізнес-процесах, які стають основою для дизайну та реалізації програмного забезпечення. Це сприяє створенню більш релевантних та цінних рішень для бізнесу.

– Структурований підхід – DDD пропонує чітку структуру та паттерни проектування, які можуть допомогти розробникам організувати код більш ефективно.

До недоліків DDD можна віднести наступні характеристики:

– Складність – DDD є досить складним підходом, що вимагає глибокого розуміння бізнес-логіки та моделювання. Це може бути важко для команд, які ще не мають досвіду з DDD.

– Часові та ресурсні витрати – процес впровадження DDD може бути довгим і вимагає значних ресурсів. Це може бути перешкодою для команд з обмеженими ресурсами або строгими часовими рамками.

– Перевантаження деталями – є ризик занадто деталізованої моделі, що може призвести до перевантаження деталями та збільшення складності системи.

2.3.4 Гексагональна архітектура

Гексагональна архітектура – це шаблон проєктування програмного забезпечення, який націлений на створення гнучкого та масштабованого програмного забезпечення шляхом ізоляції бізнес логіки додатку від коду, що відповідає за її взаємодію із сторонніми модулями системи [11].

На рис. 2.9 зображено структуру модуля, яку пропонує гексагональна архітектура. Видно, що ядром системи є така сутність як домен.

Домен – це сукупність програмного представлення сутностей, моделей, інтерфейсів та пов’язаної з ними бізнес логіки. Суть усього шаблону полягає в тому, аби максимально відокремити цю частину від зовнішніх систем. Для комунікації із зовнішніми системами домен має порти.

Порт – це контракт для використання домену, на який орієнтуютсья інші частини системи. Порти можуть описувати те, як зовнішні системи використовують бізнес логіку модуля, та те, як сам модуль використовує зовнішні модулі. Ідея введення контракту комунікаціїї із навколишнім світом також пропонується принципом інверсії залежностей із SOLID. Для того, щоб стандартизувати взаємодію сторонніх систем з портами, вводиться така сутність, як адаптери.

Адаптер – це сутність, яка є обгорткою для порта та допомагає сторонній системі підлаштувати свою структуру під роботу із ним. Будучи надбудовою над портами, адаптери також можуть описувати внутрішню та зовнішню взаємодії модуля. На рис. 2.9 видно, як сторонні системи (БД, шина повідомлень, моніторинг тощо) за допомогою адаптерів звертаються до портів ядра модуля.

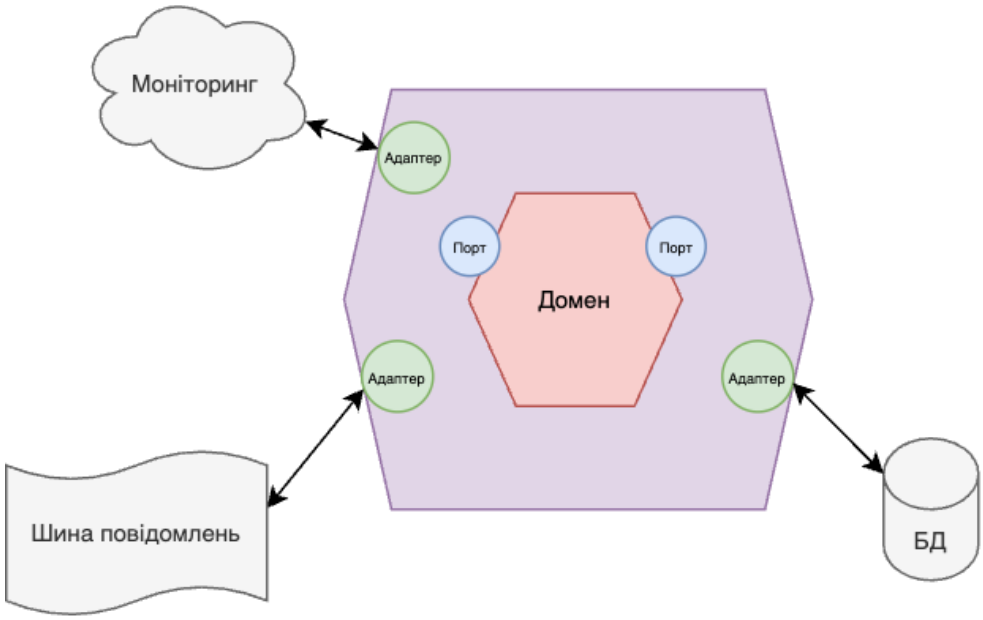

Рисунок 2.9 – Структура модуля гексагональної архітектури

Слідування гексогональній архітектурі допоможе поділити всю бізнес логіку системи на незалежні та ізольовані блоки, які можна легко замінювати між собою не порушуючи їх взаємодію. Такий підхід дає змогу досягнути також і інших переваг, що не були згадані раніше:

– Спрощення тестування – ізольовану бізнес логіку значно легше тестувати, так як вона є відносно передбачуванною. При цьому значано легше писати автоматизовані тести для такої системи, адже взаємодію із іншими модулями значно легше імітувати, коли є чітко визначений контракт.

– Простота розуміння системи – систему, що написана із використанням гексагональної архітектури, значно легше зрозуміти, через її структурованність та акцентування на уваги на бізнес логіці додатку. Це значно прискорює адаптацію нових розробників на проєкті, що розвивається.

– Простота виправлення помилок – в чітко структурованій системі легше знаходити помилки та їх виправляти. Так як модулі системи не є зв’язаними між собою, то це дає можливість швидше знаходити проблемне місце. Цьому сприятимуть і модульні тести, які стануть більш стійкими до змін внутрішньої реалізації модулів та до змін у сторонніх модулях.

Як недолік гексагональної архітектури можна виділити відстуність загального підходу до її реалізації. Існує багато реалізацій, варіацій та трактувань, що може ускладнити розуміння цього підходу.

2.3.5 Поєднання DDD підходу та гексагональної архітектури

Domain-Driven Design (DDD) та гексагональна архітектура є двома різними підходами до проектування та розробки програмного забезпечення. Ось декілька важливих відмінностей:

– Фокус – DDD зосереджений на моделюванні бізнес-домену і структурізації коду згідно з бізнес-правилами, що керують цим доменом. Гексагональна архітектура фокусується на структурізації коду таким чином, щоб ізолювати доменний шар від зовнішніх впливів і деталей інфраструктури.

– Розмежування відповідальностей – DDD використовує концепції, як-от сутності, об'єкти значення, агрегати, сервіси домену, події домену, і так далі, для розмежування відповідальностей у коді. Гексагональна архітектура використовує порти та адаптери для розмежування відповідальностей між доменним шаром та зовнішнім світом.

– Адаптація до змін – гексагональна архітектура, в основному, спрямована на адаптацію до змін, забезпечуючи можливість легкої заміни одного адаптера іншим (наприклад, заміна реляційної бази даних на NoSQL). DDD більше зосереджений на використанні відповідного архітектурного контексту, який дозволяє легко розуміти та адаптувати бізнес-правила.

Важливо зазначити, що DDD та гексагональна архітектура можуть використовуватися разом. Вони взаємодоповнюють один одного, і в багатьох випадках рекомендується їх спільне використання. Використання DDD дозволяє якісно змоделювати бізнес-домен, в той час як гексагональна архітектура допомагає ізолювати цей домен від зовнішнього світу, забезпечуючи високу тестованість та гнучкість коду [12].

Поєднання DDD та гексагональної архітектури є дуже корисним підходом до проектування та розробки програмного забезпечення. Ось деякі ключові етапи цього процесу:

– Моделювання домену – першим етапом є аналіз бізнес домену, над яким ведеться робота. Це включає спілкування з експертами домену, вивчення бізнес-правил, процесів та термінології.

– Створення доменних моделей – на основі отриманого розуміння бізнес-домену створюються доменні моделі. Це об'єкти та сутності, що представляють ключові концепції бізнесу в коді. Вони включають бізнес-правила та логіку.

– Визначення портів – наступний крок полягає в визначенні портів доменної моделі. Вони можуть включати операції отримання даних, надсилання повідомлень тощо.

– Створення адаптерів – для кожного порту мають бути створені адаптери. В рамках DDD, адаптери мають знаходитися на інфраструктурному шарі.

– Організація коду – код повинен бути організований таким чином, щоб доменна логіка була ізольована від зовнішніх технологій, тобто від шару інфраструктури.

2.4 Вимоги до інтерфейсу користувача

Проектування інтерфейсу користувача для IoT системи моніторингу стану навколишнього середовища в реальному часі вимагає зосередження на точності, оперативності та інтуїтивності. Ось деякі ключові вимоги до інтерфейсу системи:

– Відображення данних в реальному часі – інтерфейс повинен відображати дані в реальному часі, аби користувач міг бачити актуальний стан навколишнього середовища.

– Персоналізовані налаштування сповіщень – користувач повинен мати можливість встановити специфічні пороги для різних показників, при перевищенні яких він отримує сповіщення.

– Історія даних – інтерфейс повинен надавати можливість перегляду історії даних для відстеження трендів та шаблонів.

– Мобільна сумісність – оскільки IoT системи часто контролюються через мобільні пристрої, інтерфейс повинен бути оптимізований для мобільних пристроїв.

Висновки до розділу

Під час роботи над даним розділом було досліджено та проаналізовано низку аспектів, які стосуються проєктування комплексної IoT системи для моніторингу стану навколишнього середовища.

Систему було поділено на компоненти та описано їх взаємодію. Створено основні функціональні групи компонентів системи. Цей етап включав в себе аналіз взаємодії між різними компонентами, визначення їхніх ролей та відповідальностей. Можливі стани системи та прехід між ними було зображено на діаграмі станів.

Було проведено аналіз та обґрунтування вибору апаратного забезпечення системи. Сама структура апаратного забезпечення була поділена на ключові компоненти: мікроконтролер, датчик, батареї та периферіійне обладнання. Основними характеристиками, які має задовольняти апаратне забезпечення системи є універсальність, надійність, мобільність та широка розповсюдженність для легшої підтримки системи та скорішого вирішення проблем. Під описані хаарктеристики був обраний мікроконтролер ESP32 та датчик вологості та температури DHT22. Були описані основні характеристики, недоліки та переваги обраних девайсів. Для проєктування, моделювання та скриптування роботи апаратного забезпечення було використано сервіс для віртального моделювання роботи апаратного забезпечення, через те, що такий підхід значно спрощує витрати ресурсів на створення системи.

На етапі проєктування архітектури програмного забезпечення було визначено основні вимоги до архітектури системи та описано архітектурні шаблони та принципи, які будуть використані під час реалізаціїі програмного

забезпечення системи. Усе програмне забзепечення буде імплементована із використаннями принципів SOLID. Архітектурне моделюювання системи буде виконано за допомогою поєднання підходів DDD та гексагональної архітектури. Для структуризаціїї шарів системи буде використано DDD, а для ізоляції бізнес-моделей та бізнес-логіки додатку буде використано гексагональну архітектуру. Таке поєднання архітектурних принципів дозволяє створити максимально гнучку та стійку до змін систему, яка здатно швидко адаптуватися під зміни бізнес-вимог та бізнес-моделей. Незважаючи на потребу достатньо значних інвестицій, для реалізації даного підходу, в переспективі він здатен зберегти велику кількість ресурсів при подальшому розвитку та масштабуванні системи. Цей підхід також дає змогу краще тестувати програмне забезпечення, створюючи більш надійну та стабільну систему.

Фінальною частиною розділу виступило формулювання вимог до інтерфейсу користувача. Основною метою є створення зручного та легкого у використанні інтерфейсу, який відповідає усім вимогам користувача та надає можливість ефективно використовувати всю функціональність системи моніторингу.

Як результат роботи над даним розділом, була створена цілісна картинка проєкту системи, що охоплює важливі аспекти архітектури, апаратного забезпечення, принципів розробки програмного забезпечення, а також інтерфейсу користувача.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис стеку технологій

3.1.1 Мова програмування TypeScript

Використавши специфікації типів разом із традиційними функціями мови програмування JavaScript, компанія Microsoft створила мову програмування TypeScipt для веб-розробки.TypeScipt функціонує як розширення, яке компілюється в JavaScript. Це означає, що мови та технології для них є сумісними, тобто валідний код на JavaScript є також дійсним кодом TypeScript, а скомпільований TypeScript можна використати в будь-якому JavaScript проєкті. Так як JavaScript є майже безальтернативною мовою програмування у світі веб-розробки, така сумісність є дуже важливою для популяризації TypeScript. Також це дозволяє поступово інтегрувати його в існуючі проєкти.

Головною перевагою та метою створення TypeScript є доповнення сучасного JavaScript статичною типізацією. Статична типізація означає, що типи даних змінних вказуються при оголошенні, і вони не можуть бути змінені пізніше. Типи, які використовуються під час розробки на TypeScript, можуть бути як примітивними (такі як boolean, number, string тощо), так і складними (такі як інтерфейси, перелічення, кортежі тощо). Також за допомогою статичною типізації середовища розробки програмного забезпечення, такі як Visual Studio Code, WebStorm та інші, можуть надавати підсказки розробникам щодо типізації компонентів програмної системи відразу під час написання коду. Дженерики є однією із ключових особливостей TypeScript, яка додає величезну гнучкість та потужність до програмного забезпечення, дозволяючи розробникам створювати компоненти, що можуть працювати з різними типами даних, але при цьому залишатися типо-безпечними. Перелічена функціональність дозволяє значно покращити швидкість розробки системи, якісь програмного забезпечення, DX тощо. Статична типізаціє надає можливість побачити усі помилки, які виникають через типи, до того моменту, коли TypeScript код починає компілюватися в JavaScript та виконуватися.

TypeScript також надає великий набір інструментів для ООП, таких як класи, інтерфейси та модулі. Це спрощує створення великих і складних програмних систем на JavaScript.

Також TypeScript додає нові функції, які ще не були впроваджені в стандарт JavaScript, але передбачається їхнє впровадження в майбутніх версіях, такі як, наприклад, декоратори. Декоратори – це спеціальний тип оголошення, який може бути приєднаний до класів, методів, аксесорів та властивостей. Вони можуть бути використані для модифікації поведінки або додавання нової поведінки. Це є основою для таких функцій, як анотації в Angular.

TypeScript має вбудовану підтримку JSX, що робить його відмінним вибором для роботи з бібліотеками та фреймворками, такими як React. JSX - це синтаксис, який дозволяє вам писати HTML-подібний код безпосередньо всередині JavaScript (або TypeScript).

До недоліків TypeScript можна віднести наступні характеристики:

– Компіляція – TypeScript повинен бути компільований в JavaScript. Це може зайняти додатковий час і стати причиною затримки в процесі розробки, особливо великих проектів.

– Сумісність з бібліотеками – хоча TypeScript і сумісний з JavaScript, деякі бібліотеки JavaScript можуть не мати відповідних описів типів. Це може ускладнити процес розробки із використанням цих бібліотек.

Підсумовуючи, TypeScript пропонує ряд переваг для розробників JavaScript. Він вносить статичну типізацію, яка може допомогти виявити помилки раніше та покращити якість коду. Він додає багато функцій, які допомагають організувати та структурувати код. І, нарешті, він забезпечує сумісність з JavaScript, що спрощує перехід на TypeScript для існуючих проектів. Особливо добре TypeScript підходить для великих проєктів, так як наявність типів значно спрощує та прискорює інтеграцію нових розробників у процес створення та розвитку програмної системи, роблячи структуру та архітектуру системи більш виразною та зрозумілою.

3.1.2 Платформа NodeJs

NodeJs – це вільне та відкрите середовище виконання для JavaScript, побудоване на ядрі V8 від Google Chrome [13]. Мета створення NodeJs полягає в тому, щоб створити платформу, яка здатна обробляти велику кількість одночасних з'єднань. Для такої мети було обрано JavaScript через його подійно-орієнтовану модель і використано двигун V8 від Google Chrome через його швидкість та потужність. Це дозволило зробити одну із найпопулярніших мов програмування JavaScript, яка виконувалася лише в браузері, універсальною та ще більш поширеною. Дана технологія надає можливість використовувати JavaScript в будь-якій сфері розробки програмного забезпечення. У світіт веб-розробки, така універсальність є важливою перевагою, адже програмна система, клієнтська та серверна частини якої створені за допомогою однієї мови програмування, є значно універсальнішою та простішою для розуміння ніж система, яка написана різними мовами програмування.

Основою NodeJs є концепція подійно-орієнтованого програмування. Код в Node.js виконується асинхронно, що означає, що виконання коду не блокується, коли відбуваються операції вводу-виводу (I/O), такі як читання з файлової системи, звернення до бази даних або відправлення HTTP-запитів. Замість того, функції зворотного виклику (callback) використовуються для обробки результатів цих операцій, коли вони стануть доступними. Це дозволяє Node.js ефективно обробляти велику кількість одночасних з'єднань без необхідності використовувати багато потоків, як це роблять багато інших мов програмування. На рис. 3.1 зображено модель роботи NodeJs. Вхідні запити, що надходять у систему відразу потрапляються до черги подій, де вони зберігаються. Звідти події потрапляють до нескінченного циклу подій, за допомогою якого працює JavaScript. Як тільки цикл подій натрапляє на блокуючу задачу вводу-виводу, ця задача передається до одного із робочих потоків, що містяться в ОС. Під час виконання цих завдань робочими тредами ОС, цикл подій приймає нові події. Коли робочий тред ОС звершує

виконання задачі вводу-виводу із певним результатом, функція, що оброблятиме цей результат стає в чергу для потрапляння в цикл подій.

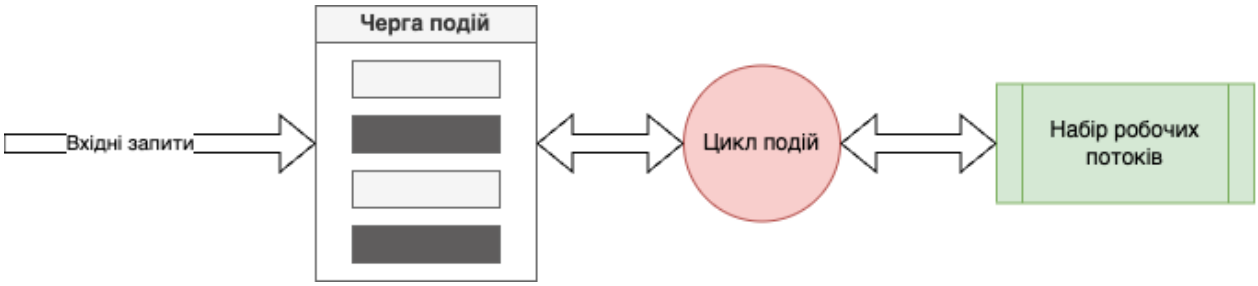

Рисунок 3.1 – Подійно-орієнтована модель роботи NodeJs

Описана модель роботи має вразливі місця при роботі із складними для процесору JavaScript завданнями, які можуть блокувати роботу циклу подій. Ці завдання не можна перенести в робочі треди ОС, як це відбувається із задачами вводу-виводу. Для цього, NodeJs має власні робочі треди, в яких можна виконувати такі задачі, не блокуючи основний цикл подій.

NodeJs має багату екосистему модулів, що базується в менеджері пакетів NPM (Node Package Manager). Це означає, що розробник може легко додавати сторонню функціональність до свого застосунку, використовуючи вже написаний та протестований код. NPM – це найбільший репозиторій пакетів програмного забезпечення у світі, і він включає модулі для роботи з мережею, файловою системою, базами даних, автентифікацією, тестуванням тощо.

Важливо зазначити, що, хоча Node.js і використовує JavaScript, воно містить декілька розширень та особливостей, які відрізняють його від традиційного JavaScript, що використовується в браузерах. Наприклад, воно містить модульну систему (CommonJS), яка дозволяє імпортувати та експортувати код між різними файлами, а також API для роботи з файловою системою, мережами, потоками та іншими можливостями сервера.

NodeJs широко використовується для створення веб-серверів, API, інструментів командного рядка та інших типів застосунків.

Підсумовуючи, Node.js – це потужне та гнучке середовище для створення різноманітних застосунків на основі JavaScript. Його модель подійно-орієнтованого програмування, багата екосистема модулів та базування на JavaScript робить його привабливим вибором для багатьох проектів, більшість з яких орієнтуються на галузь веб-розробки.

3.1.3 СУБД PostgreSQL

PostgreSQL – є досить поширеною об'єктно-реляційною системою управління базами даних, серед основних характеристик якої можна назвати її надійність та висока масштабованність [14].

Заснований на моделі реляційної бази даних, PostgreSQL дотримується високого стандарту ACID (Atomicity, Consistency, Isolation, Durability - Атомарність, Согласованість, Ізоляція, Стійкість), що гарантує надійність і стабільність транзакцій. Ця міцна підтримка ACID робить PostgreSQL ідеальним вибором для великомасштабованих застосунків, які працюють із високим навантаженням.

PostgreSQL має вбудовану підтримку об’єктів, які дозволяють розробникам створювати свої власні типи даних, спрощуючи взаємодію зі складними структурами даних.

PostgreSQL пропонує користувачам широкі можливості для розширювання існуючої системи. Розширювання у PostgreSQL – це особливі пакети, які можуть бути встановлені в базу даних для додавання додаткових функцій, типів даних, операторів, функцій агрегування, індексаторів тощо. Завдяки розширенням PostgreSQL стає надзвичайно гнучким та адаптивним інструментом, який може бути налаштований під конкретні потреби.

PostgreSQL обладнаний повноцінною підтримкою SQL, включаючи підтримку вкладених транзакцій, з'єднань, підзапитів, тригерів та збережених процедур. PostgreSQL пропонує вбудовані функції, які можуть здіснювати маніпулювання даними, спрощуючи роботи із системою та даними.

PostgreSQL має підтримку декількох способів масштабування системи, а саме вертикального та горизонтального.

Вертикальне масштабування полягає у збільшенні потужності одного сервера. Це може включати додавання більше центральних обробних пристроїв, оперативної пам'яті, місця для зберігання даних або кращих мережевих адаптерів до існуючої машини. Переваги вертикального масштабування включають збереження простоти архітектури, оскільки користувачу не потрібно розробляти додаткову логіку для розподілу даних та запитів між різними серверами. Однак, вертикальне масштабування може здійснюватися лише до тих пір, поки не буде обмежено потужностями сервера або занадто великою ціною підтримки системи.

Горизонтальне масштабування полягає у додаванні більше серверів до пулу. В контексті баз даних, це може означати розподіл даних між кількома серверами (шардінг) або реплікацію даних між серверами для забезпечення високої доступності. Горизонтальне масштабування дозволяє розробнику додавати обладнання відповідно до потреб, що робить цей метод гнучкішим та більш ефективним за вартістю при великих обсягах даних. Але цей спосіб масштабування потребує більшої кількості часу на реалізацію, так як вона включає в себе великий обсяг із планування, розподілення та управління навантаженням.

PostgreSQL має високу стійкість до відмов, завдяки використанню журналу відновлення та реплікації даних. Це забезпечує надійне відновлення даних у випадку відмови або втрати даних.

До недоліків PostgreSQL можна віднести наступні характеристики:

– Складність налаштування – PostgreSQL є складнішим для налаштування та адміністрування в порівнянні з іншими СУБД, і хоча велика кількість різних налаштувань додає гнучкості, це може стати викликом для новачків або для тих, хто не має глибоких знань про бази даних.

– Ресурсоємкість – PostgreSQL вимагає більше системних ресурсів у порівнянні з деякими іншими СУБД, такими як, наприклад, MySQL. Це може бути проблемою для систем з обмеженими ресурсами.

Загалом, PostgreSQL є потужною, надійною та гнучкою СУБД, яка відповідає потребам великої кількості застосунків.

3.1.4 React

React – це бібліотека для JavaScript від компанії Meta, що призначена для створення користувацьких інтерфейсів. React використовується для побудови сучасних, швидких веб-інтерфейсів із великим рівнем інтерактивності [15].

До ключових характеристик React можна віднести:

– Компонентний підхід – React структурує інтерфейси користувача у формі компонентів. Компонент можна уявити як самостійний фрагмент коду, що містить в собі HTML-розмітку і відповідну логіку. Компоненти можуть включати інші компоненти, що дозволяє створювати складні ієрархії. Крім того, вони можуть бути повторно використовувані в межах одного застосунку, що підвищує ефективність розробки. Основною перевагою компонентного підходу є декомпозиція клієнтського коду, яка допомагаж зменшити ментальне навантаження розробника, що працює над програмним забезпеченням.

– Віртуальний DOM – це концепція, за якою робить рендеринг користувацького інтерфейсу надзвичайно ефективним за ресурсами та швидкістю роботи. Віртуальний дом являє собою копію реального DOM-дерева, яке створює браузер. Він вирішує проблему зайвого рендерингу DOM-елементів. Наприклад, коли на сторінці сайту змінюється якась частина інтерфейсу, React, за допомогою віртуального DOM перерендерить не усю сторінку, а тільки змінену частину інтерфейсу. На рис. 3.2 зображено алгоритм роботи віртуального DOM. Після взаємодії користувача, компонент змінює лише віртуальний DOM. Далі, за допомогою внутрішніх алгоритмів, React, порівнює віртуальний DOM із реальним, та знаходить місця, що змінились, і рендерить тільки їх, змінюючи реальний DOM. Для того, щоб алгоритм порівняння працював ефективно, він використовує багато різних оптимізацій. Основними є два правила, які враховуються при обході віртуального DOM дерева: два елементи з різними типами, дають різні DOM-

дерева, за допомогою атрибуту key розробник може помітити стабільні елементи. Якщо по одноми із правил React визначає, що один дерево компонентів могло змінитися, воно перестає перевірятися вглиб, демонтується, та вставляються в реальний DOM вже як нові елементи.

– JSX – це синтаксис, що дозволяє включати HTML-подібні конструкції безпосередньо в код JavaScript. Це значно спрощує процес написання компонентів, оскільки розробники можуть використовувати звичний HTML-синтаксис для створення елементів інтерфейсу. Крім того, JSX дозволяє вбудовувати вирази JavaScript безпосередньо в розмітку.

– Стан та властивості – React використовує два ключові концепти для управління даними в компонентах: стан (state) та властивості (props). Стан – це об'єкт, що містить дані, які можуть змінюватися в процесі життєвого циклу компонента. Коли стан компонента змінюється, компонент перерендерюється, що відображає ці зміни в інтерфейсі. Властивості – це дані, що передаються в компонент від його батьківського компонента. Вони дозволяють батьківському компоненту керувати поведінкою та відображенням дочірнього компонента.

– Однонаправлені потоки даних – в React дані потікують від батьків до дітей через властивості. Цей підхід забезпечує більшу прозорість і контроль над тим, як дані передаються та використовуються в застосунку.

– Контекст – контекст в React дозволяє передавати дані через дерево компонентів без необхідності передачі властивостей на кожному рівні. Це може бути корисним для даних, які мають бути доступними для багатьох компонентів на різних рівнях ієрархії.

– Контрольовані компоненти – React підтримує концепцію контрольованих компонентів, де стан форми зберігається в стані компонента і оновлюється за допомогою функцій обробки подій. Це надає повний контроль над полями форми та виключає неочікувані зміни введених даних.

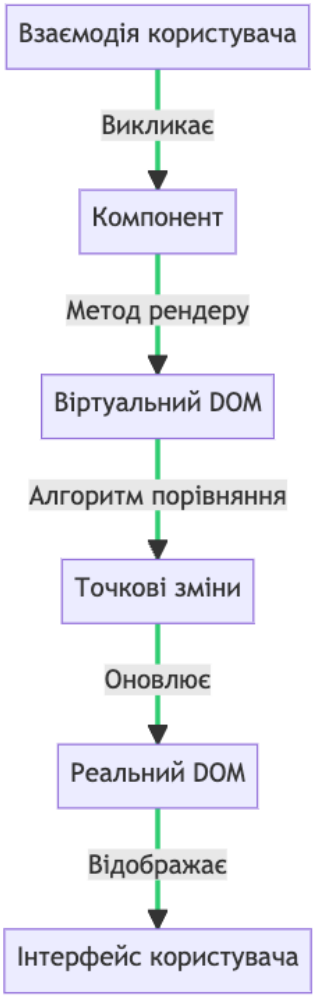

Рисунок 3.2 – Алгоритм роботи віртуального DOM

3.2 Скриптування апаратного забезпечення додатку

Для того, щоб запрограмувати мікроконтролер ESP32 та датчик DHT22 необхідно скористатися варіацією мови C++, яка називається Arduino C++ або Arduino Language та використовується для програмування мікроконтролерів [16]. Основна різниця полягає в тому, що програми Arduino розбиваються на дві основні функції: setup() та loop():

– Функція setup() викликається один раз при запуску програми і зазвичай використовується для ініціалізації змінних, налаштування режимів виводу пінів, початку зв'язку з іншими пристроями та інших завдань, що виконуються один раз.

– Функція loop() викликається безкінечно після завершення setup(). Тут відбувається більшість роботи: зчитування датчиків, управління пристроями, взаємодія з користувачем тощо.

На рис. 3.3 зображено приклад найпростішого скрипта під обране апаратне забезпечення із використанням, що демонструє роботу функцій setup та loop. Цей скрипт пфдключається до датчику та виводить значення актуальні значення вологості та температури щосекунди.

```
#include "DHTesp.h"

const int DHT_PIN = 15;

DHTesp dhtSensor;

void setup() {
  Serial.begin(115200);
  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);
}

void loop() {
  TempAndHumidity  data = dhtSensor.getTempAndHumidity();
  Serial.println("Температура: " + String(data.temperature, 2) + "°C");
  Serial.println("Вологість: " + String(data.humidity, 1) + "%");
  Serial.println("---");
  delay(1000);
}
```

Рисунок 3.3 – Виведення актуальних показників навколишнього середовища

На рис 3.4 виведено результат роботи попереднього скрипта. Як видно, система запущена протягом 10 секунд, а в консолі присутні значення температури та вологості.

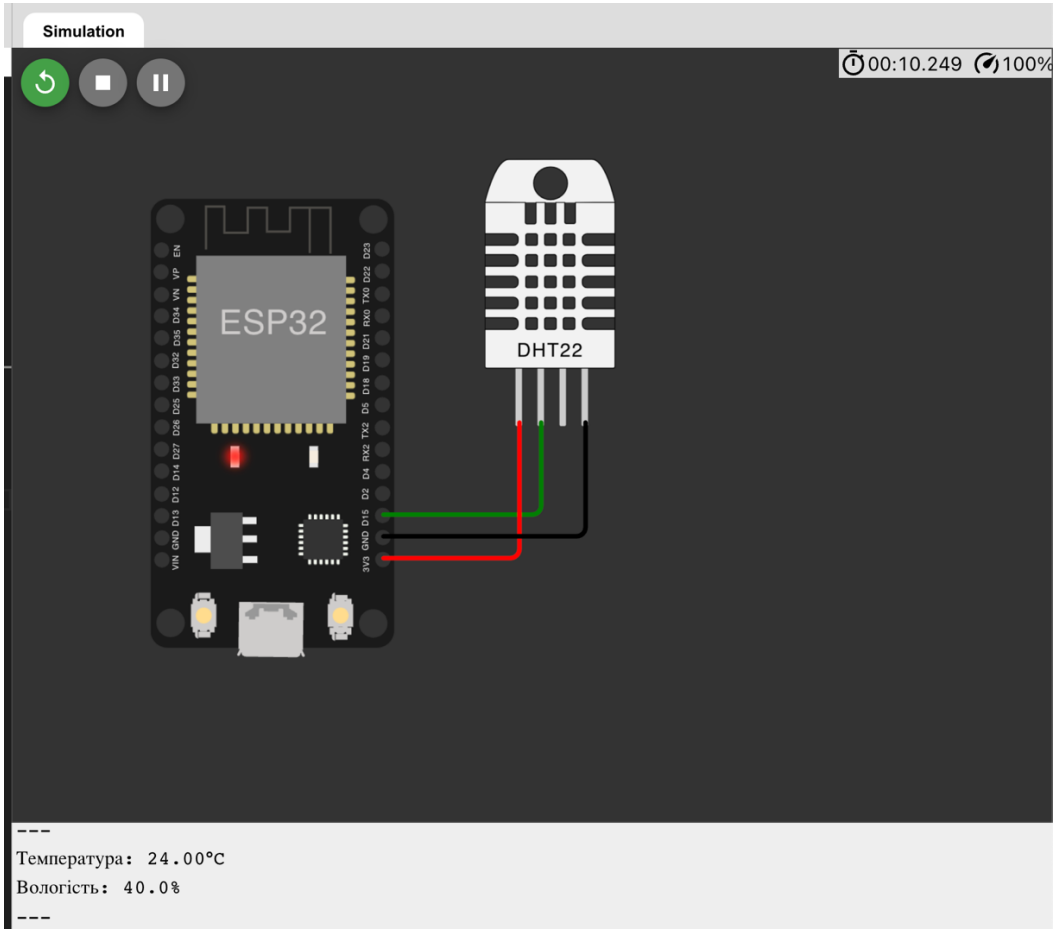

Рисунок 3.4 – Результат роботи скрипта на рис. 3.3

Для IoT системи моніторингу необхідно написати скрипт, який відправлятиме щосекунди актуальні значення стану навколишнього середовища на сервер, який обробить та надасть данні до клієнтської частини додатку. Першим етапом розробки скрипту є імпорт бібліотек (рис. 3.5).

```
1  #include <WiFi.h>
2  #include <PubSubClient.h>
3  #include <ArduinoJson.h>
4  #include <DHT.h>
```

Рисунок 3.5 – Імпорт бібліотек

Далі необхідно визначити пін, до якого підключиться датчик DHT22 та його тип (рис. 3.6).

```
#define DHTPIN 4
#define DHTTYPE DHT22
```

Рисунок 3.6 – Визначення порту підключення та типу датчику

Далі виконується ініціалізація ПЗ для роботи із датчиком (рис. 3.7).

```
DHT dht(DHTPIN, DHTTYPE);
```

Рисунок 3.7 – Ініціалізація ПЗ для роботи із датчиком

Далі визначаються SSID та пароль Wi-Fi мережі, а також URL сервера та порт для підключення через MQTT (рис. 3.8).

```
const char* ssid = "your_ssid";
const char* password =  "your_password";
const char* mqttServer = "your-server.com";
const int mqttPort = 1883;
```

Рисунок 3.8 – Визначення параметрів підключення до мережі та сервера

Далі виконується ініціалізація MQTT, який використовується для з'єднання з сервером та обміну даними (рис. 3.9).

```
WiFiClient espClient;
PubSubClient client(espClient);
```

Рисунок 3.9 – Ініціалізація MQTT

В функції setup ініціалізується датчик DHT22, підключення до Wi-Fi та MQTT (рис. 3.10).

```
void setup() {
  Serial.begin(115200);
  delay(1000);

  dht.begin();

  WiFi.begin(ssid, password);

  while (WiFi.status() ≠ WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi ..");
  }
  Serial.println("Connected to the WiFi network");

  client.setServer(mqttServer, mqttPort);

  while (!client.connected()) {
    Serial.println("Connecting to MQTT ... ");

    if (client.connect("ESP32Client")) {
      Serial.println("connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
```

Рисунок 3.10 – Ініціалізація та налаштування системи

Функція loop обробляє всі події WebSocket, читає дані з сенсора DHT22, формує JSON-об'єкт із акоуальними даними про стан навколишнього середовища ти відправляє його через WebSocket (рис. 3.11).

```
void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

    DynamicJsonDocument doc(1024);
    doc["temperature"] = t;
    doc["humidity"] = h;
    String jsonString;
    serializeJson(doc, jsonString);

    client.publish("esp32/data", jsonString.c_str());
  } else {
    Serial.println("WiFi Disconnected");
  }

  delay(1000);
}
```

Рисунок 3.11 – Основний цикл системи

3.3 Розробка серверної частини додатку

Розробка серверної частини додатку проходила за допомогою NodeJs та мови TypeScript. Так як JavaScript та є мультипарадигменними мовами програмування, то вирішено було притримуватися цього підходу до розробки програмного забезпечення, аби мати змогу використати більше доступних переваг цих мов.

Для розробки програмного забезпечення системи було обрано підхід із використанням гексагональної архітектури та DDD. Аби відтворити даний підхід, було створено папки в проєкті, які зображені на рис. 3.12.

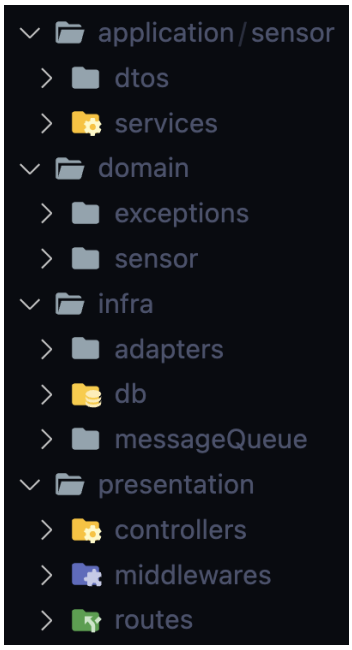

Рисунок 3.12 – Основні структурні компоненти проєкту

На рис. 3.12 відображено наступні структурні компоненти системи:

– application – шар містить бізнес-процеси та координацію роботи. Це включає в себе обробку даних від входів, переклад даних в доменні об'єкти, виклик методів доменного шару, збереження результатів обчислень, а також надсилання вихідних даних.

– domain – центральний шар, в якому визначаються доменні сутності, їх властивості та методи, а також взаємодія між ними.

– infrastructure – шар, в якому реалізовані технічні деталі та вся інфраструктура.

– presentation – шар, що відповідає за відображення даних користувачу та взаємодію з ним.

Для того, щоб спростити розгортання серверу, було використано бібліотеку Express, що є одним із найпопулярніших інструментів для розгортання веб-серверів в екосистемі NodeJs. Express надає розширену систему маршрутизації, яка дозволяє визначати маршрути і обробники для HTTP-запитів. Express.js надає засоби для відправлення відповідей на клієнтські запити, включаючи відправлення JSON, HTML, текстових файлів, статусних кодів тощо.

Серверна частина додатку буде слідкувати за змінами, які спостерігають датчики, та надсилати їх на клієнтську частину додатку за допомогою протоколу WebSocket, який дозволяє встановити двосторонню комунікацію між клієнтом та сервером. Зміни від пристроїв IoTнадходитимуть до серверу через протокол MQTT. Для цього сервер ініціалізує клієнт MQTT за допомогою бібліотеки mqtt. Для того, щоб клієнти MQTT на сервері та на мікроконтролері могли обмінюватися одне із одним повідомленнями про стан навколишнього середовища потрібен сервер-брокер, через який відбуватиметься взаємодія. В якості брокера використано безкоштовний сервіс EMQ X. Взаємодія відбуватиметься наступним чином: мікроконтролер виступатиме MQTT-клієнтом, що відправляє повідомлення (дані про температуру та вологість) до MQTT брокера (EMQ X). З іншого боку, сервер також буде виступати в ролі MQTT клієнта, що підписується на ці повідомлення через брокера. Коли мікроконтролер відправляє повідомлення до брокера, брокер автоматично передає це повідомлення всім клієнтам, що підписалися на відповідну тему. Так сервер зможе приймати та обробляти ці повідомлення. Ця модель публікації та підписки дозволяє масштабувати систему, додавати нові пристрої, та ефективно розподіляти повідомлення між різними клієнтами.

3.4 Розробка бази даних

На основі огляду предметної області та створення вимог до роботи системи створено наступний склад таблиць БД:

– USER (Користувач) – це таблиця, яка містить інформацію про користувачів системи моніторингу. У ній зберігається їхнє ім'я, електронна пошта та основний ідентифікатор (user_id). Кожен користувач може мати свій користувацький профіль (USER_PROFILE). USER_PROFILE (Профіль користувача) – це таблиця, в якій зберігається інформація про налаштування та вподобання користувача. Кожен профіль має унікальний ідентифікатор (profile_id) та зв'язок з користувачем за допомогою зовнішнього ключа (user_id).

– DEVICE (Пристрій) – це таблиця, що містить дані про IoT пристрої, які використовуються у системі. Вона містить ідентифікатор пристрою (який є первинним ключем), місцезнаходження, тип пристрою (DEVICE_TYPE) та зв'язок з користувачем за допомогою зовнішнього ключа (user_id).

– DEVICE_TYPE (Тип пристрою) – це таблиця, в якій зберігається інформація про різні типи пристроїв, які можуть бути використані у системі. Кожен тип пристрою має свій унікальний ідентифікатор (device_type_id), назву та опис.

– SENSOR (Датчик) – це таблиця, що містить дані про датчики, які використовуються в пристроях системи. Вона містить ідентифікатор датчика (який є первинним ключем), статус датчика, тип датчика (SENSOR_TYPE) та зв'язок з пристроєм за допомогою зовнішнього ключа (device_id).

– SENSOR_TYPE (Тип датчика) – це таблиця, в якій зберігається інформація про різні типи датчиків, які можуть бути використані у системі. Кожен тип датчика має свій унікальний ідентифікатор (sensor_type_id), назву та опис.

– DATA (Дані) - це таблиця, в якій зберігаються дані, які збираються датчиками. Вона містить часову мітку, значення даних, унікальний ідентифікатор (data_id) та зв'язок з датчиком за допомогою зовнішнього ключа (sensor_id).

– ALERT (Тривога) – це таблиця, в якій зберігаються повідомлення про тривоги, які отримує користувач. Кожна тривога має свій унікальний ідентифікатор (alert_id), рівень тривоги, тип тривоги (ALERT_TYPE) та зв'язок з даними за допомогою зовнішнього ключа (data_id).

– ALERT_TYPE (Тип тривоги) – це таблиця, в якій зберігається інформація про різні типи тривог, які можуть бути відправлені у системі. Кожен тип тривоги має свій унікальний ідентифікатор (alert_type_id), назву та опис.

Описана структура БД дозволяє зберігати і організовувати дані про користувачів, пристрої, датчики, дані та тривоги у системі моніторингу стану навколишнього середовища. Кожна таблиця має свої поля та взаємозв'язки з іншими таблицями, що допомагає забезпечити цілісність даних та зручну роботу з БД.

3.5 Розробка клієнтської частини додатку

Для написання клієнтської частини додатку було використано React. Для зберігання стану додатку було використано популярну бібліотеку для зберігання стану Redux. Взагалі, Redux не залежить від стеку та може бути використаним із різними технологіями в інфраструктурі JavaScript, але найбільше його використовуються саме у зв’язці з React. Вона є особливо корисною для додатків з великою кількістю станів, які потребують управління. Основна концепція Redux полягає в тому, що вся інформація, яку використовує додаток, зберігається в єдиному об’єкті сховищі [17]. Сховище виступає єдиною точкою доступу до даних, на яку покладаються усі елементи системи. Це допомагає уникнути проблем із синхронізацією стану у всіх частинах додатку. Redux заснований на наступних принципах:

– Об’єкт стану доступний лише для читання – єдиний спосіб змінити стан полягає в тому, щоб створити об'єкт, що описує, що сталося. Це забезпечує консистентність стану.

– Зміни стану виконуються за допомогою чистих функцій – ці функції приймають об’єкти, що описують зміну стану, та повертають новий об’єкт стану. Функція вважається чистою, якщо вона не змінює вхідні параметри та інші глобальні сутності.

Головна сторінка додатку матиме декілька важливих віджетів, за допомогою яких користувач може взаємодіяти із системою. На рис. 3.14 зображено віджети для перемикання стану моніторингу параметрів температури та вологості.

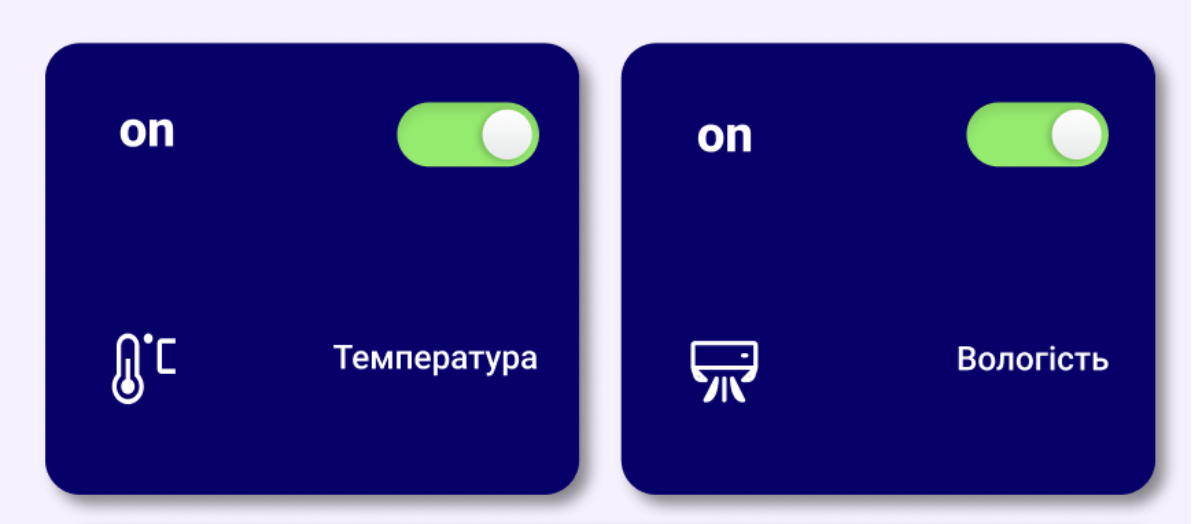

Рисунок 3.13 – Віджети для перемикання стану моніторингу

На рис. 3.14 зображено віджет, який містить список повідомлень, які система надсилає користувачу, при погіршені або нормалізації стану середовища.

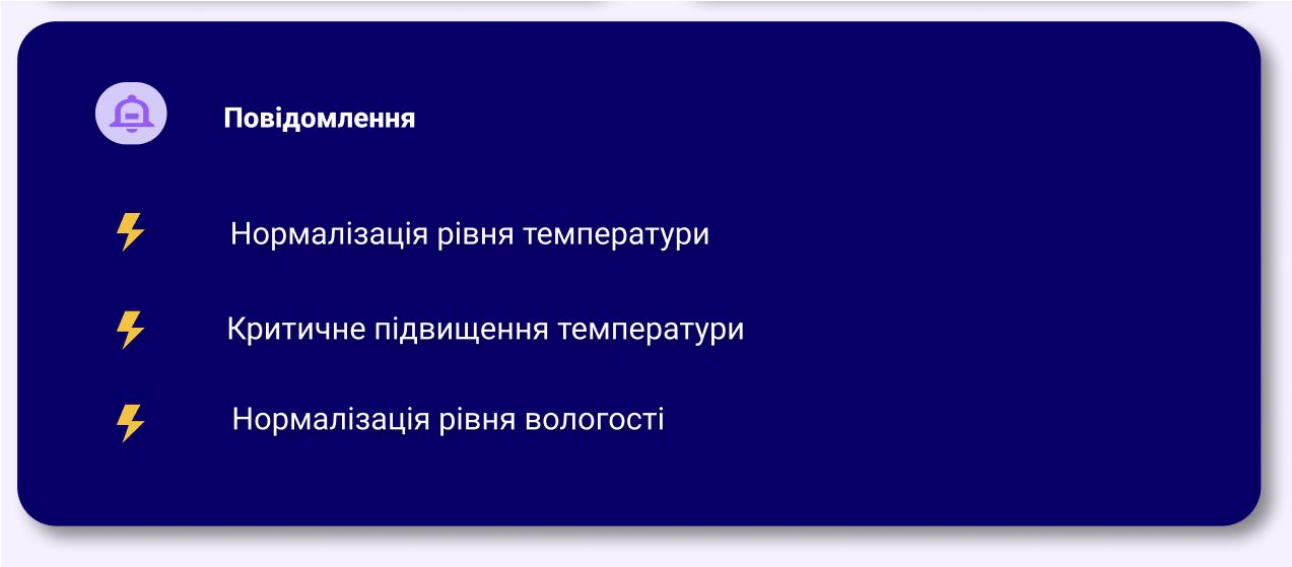

Рисунок 3.14 – Віджет для перегляду повідомлень системи

На рис 3.15 зображено віджети, які відображають актуальні значення стану навколишнього середовища, що надходять від датчиків. Ці значення предаються із датчиків на сервер в реальному часі і, якщо нові значення відрізняються від попередніх, сервер відправляє оновлені дані до клієнтської частини, яка відображає ці дані на віджетах.

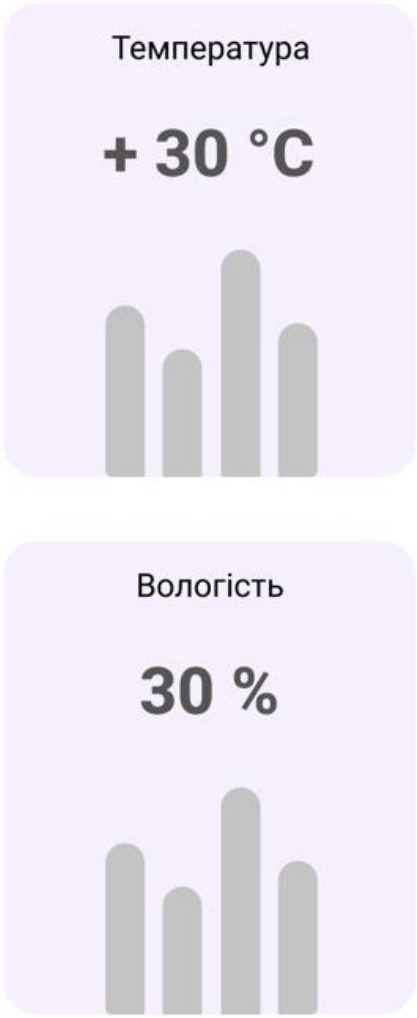

Рисунок 3.15 – Віджет для перегляду показників датчиків

На рис. 3.16 зображено сторінку із віджетами.

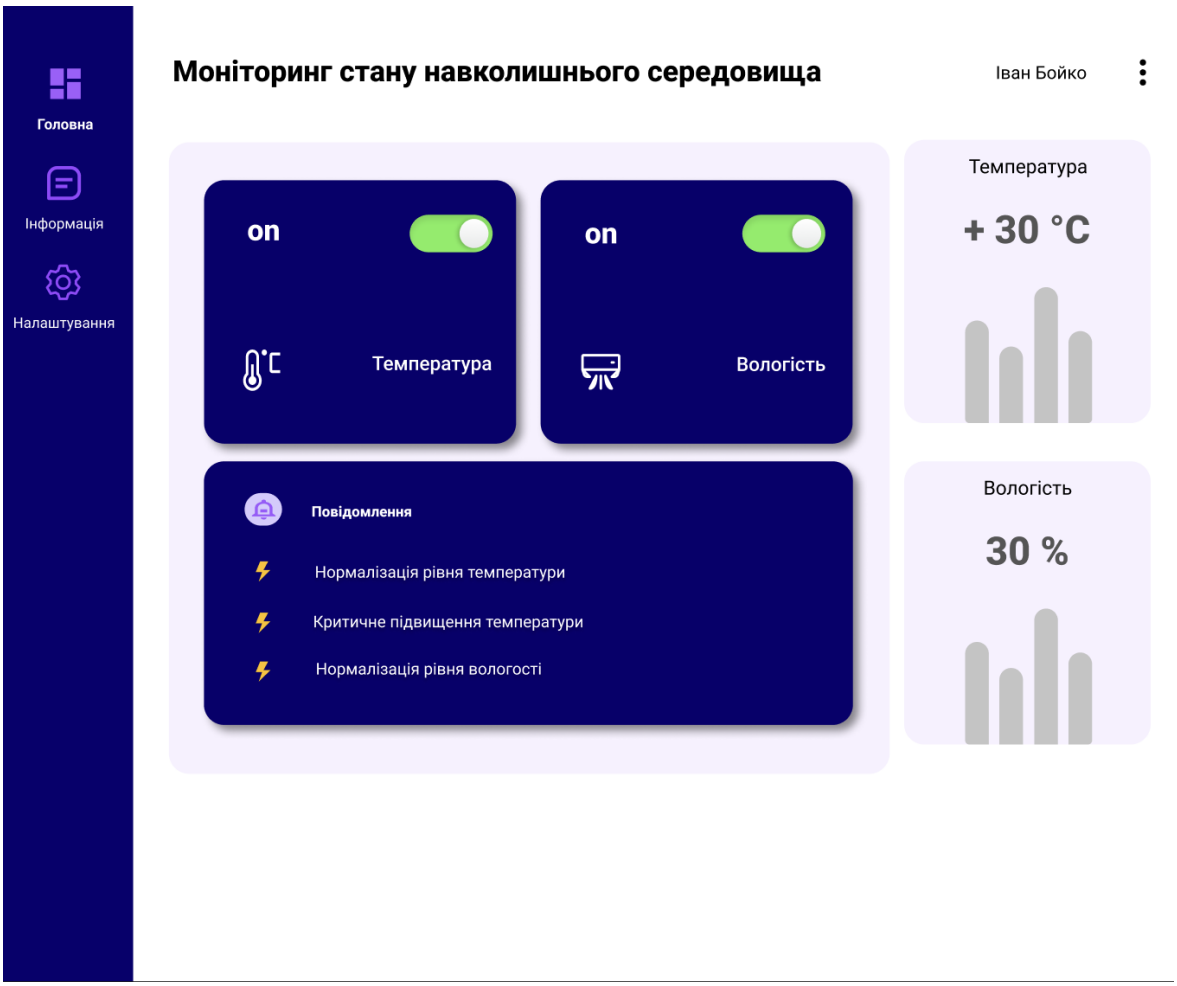

Рисунок 3.16 – Загальний вигляд сторінки із віджетами

На екрані налаштувань користувач системи, може встановити граничні значення показників, при перевищенні яких, система надішле повідомлення користувачу (рис. 3.17).

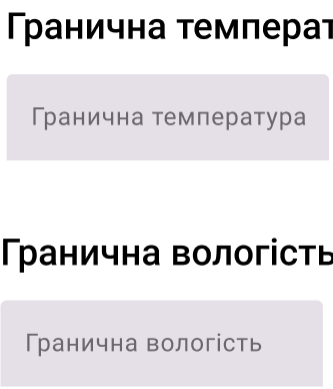

Рисунок 3.17 – Налаштування граничних значень показників

На рис. 3.18 зображено інтерфейс сторінки налаштувань.

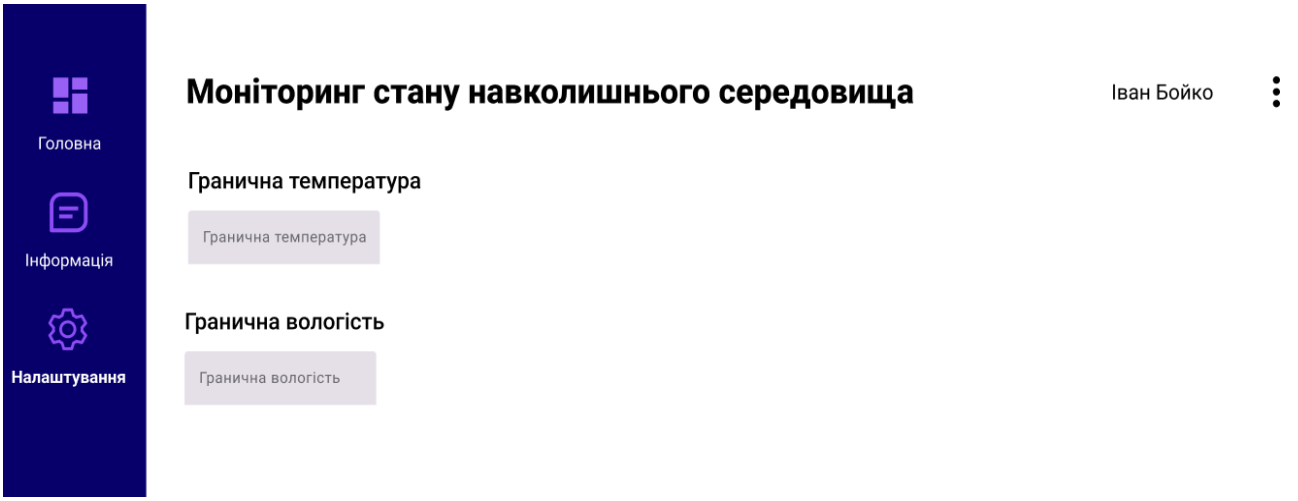

Рисунок 3.18 – Інтерфейс сторінки налаштувань

Висновки до розділу

Під час роботи над даним розділом було проаналізовано та обрано інструменти для розробки програмного забезпечення системи, такі як мови програмування, платформи, бібліотеки, протоколи тощо. Були описані переваги та недоліки технологій, їх основні характеристики та області використання.

За допомогою обраних технологій було реалізовано програмне забезпечення із дотриманням завчасно підготованої та спроєктованої архітектури. Відповідно до вимог системи та результатів дослідження предметної області була розроблена структура БД.

ВИСНОВКИ

В даній дипломній роботі було проведено детальне дослідження та розроблено IoT систему для моніторингу стану навколишнього середовища в реальному часі.

Спочатку було проведено загальний огляд об'єкта дослідження, що включає аналіз важливості моніторингу стану навколишнього середовища. Також було здійснено аналіз і порівняння існуючих систем, таких як Aclima Sensing Network, IBM Green Horizons, Arable Mark 2, Libelium Smart Water та Cisco Kinetic for Cities. Кожна з цих систем має свої переваги та недоліки. Порівняльний аналіз дав змогу краще оцінити ці системи, та сформувати вимоги до власної.

Далі було створено загальну схему взаємодії компонентів системи. Тут було визначено необхідне апаратне забезпечення, зокрема мікроконтролер ESP32 та датчик DHT22, які використовуються для збору даних про стан навколишнього середовища. Проведено аналіз обраного апаратного забезпечення, його особливості та можливості. Також було обрано середовище для проєктування апаратного забезпечення системи, включаючи вибір необхідних компонентів та з'єднання між ними.

Був відображений процес проєктування архітектури програмного забезпечення системи. Були проаналізовані та описані принципи створення програмного забезпечення SOLID, а також DDD та гексагональної архітектури. Ці підходи до розробки дозволяють забезпечити гнучкість, масштабованість та підтримку доменної моделі системи. Архітектура була створена за допомогою поєднання підходу DDD та гексагональної архітектури для забезпечення кращої організації коду та розмежування шарів системи.

Були визначені вимоги до інтерфейсу користувача, що описують функціональні та нефункціональні вимоги до системи. Вимоги до інтерфейсу користувача дозволяють забезпечити зручність взаємодії користувача з системою та забезпечити доступ до необхідної інформації.

Були описані використані технології. Було розглянуто мову програмування TypeScript, яка дозволяє розробляти типобезпечні застосунки для веб-серверів та веб-додатків. Також була описана платформа NodeJs, яка забезпечує середовище для виконання JavaScript на сервері. Для зберігання даних була використана СУБД PostgreSQL, а для розробки клієнтської частини додатку використовувалася бібліотека React. Було описано та ілюстровано процес розробки системи.

Розроблена IoT система для моніторингу стану навколишнього середовища в реальному часі успішно виконує свої функції. Ця система має потенціал для використання у різних галузях, де важлива якість повітря, температура, рівень шуму та інші параметри навколишнього середовища. Застосування такої системи може сприяти охороні навколишнього середовища, ефективному плануванню міських просторів та оптимізації промислових процесів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Birinderjit Singh. IoT based greenhouse monitoring and control system. Kindle Edition. 2021. 69 p.
2. В. М. Боголюбов, М. О. Клименко, В. Б. Мокін. «Моніторинг довкілля». Вінниця: ВНТУ, 2010. 232 с.
3. Alice James, Avishkar Seth, Subhas Chandra Mukhopadhyay. IoT System Design: Project Based Approach (Smart Sensors, Measurement and Instrumentation Book 41). Springer. 2021. 441 p.
4. Chet Hosmer. Defending IoT Infrastructures with the Raspberry Pi: Monitoring and Detecting Nefarious Behavior in Real Time. Apress. 2018. 198 p.
5. Документація LPWAN. URL: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lpwan-technologies/ (дата звернення: 07.06.2023).
6. Документація MQTT. URL: https://mqtt.org/ (дата звернення: 07.06.2023).
7. Документація ESP32. URL: https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/index.html (дата звернення: 07.06.2023).
8. Документація DHT22. URL: https://cityos-air.readme.io/docs/4-dht22-digital-temperature-humidity-sensor (дата звернення: 07.06.2023).
9. Бранець І. Чому SOLID – важлива складова мислення програміста. Розбираємося на прикладах з кодом. Dou. URL: https://dou.ua/lenta/articles/solid-principles/ (дата звернення: 07.06.2023).
10. Кущ Б. Що таке Domain-Driven Design та на якому етапі варто його впроваджувати в продукт. Dou. URL: https://dou.ua/forums/topic/39874/ (дата звернення: 07.06.2023).
11. Martinez P. Hexagonal Architecture, there are always two sides to every story. Medium. URL: https://medium.com/ssense-tech/hexagonal-architecture-there-are-always-two-sides-to-every-story-bc0780ed7d9c (дата звернення: 07.06.2023).

12.DDD, Hexagonal, Onion, Clean, CQRS, … How I put it all together. URL: https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/ (дата звернення: 07.06.2023).

13.Документація Node.js. URL: https://nodejs.org/en/docs (дата звернення: 07.06.2023).

14.Документація PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 07.06.2023).

15.Документація React. URL: https://react.dev/ (дата звернення: 07.06.2023).

16.Christopher Michael Kormanyos. Real-Time C++: Efficient Object-Oriented and Template Microcontroller Programming. Springer. 2021. 551 p.

17.Документація Redux. URL: https://redux.js.org/ (дата звернення: 07.06.2023).

18.Bauer H., Patel M., Veira J. Internet of Things: Opportunities and challenges for semiconductor companies. McKinsey & Company. URL: https://www.mckinsey.com/industries/semiconductors/our insights/internet-of-things-opportunities-and-challenges-for-semiconductor companies (дата звернення: 07.06.2023).

19.Hexagonal Architecture Design Pattern | Mitrais Blog. Mitrais: A World Class Software Development Company. URL: https://www.mitrais.com/news-updates/hexagonal-architecture-design-pattern/ (дата звернення: 07.06.2023).

20.Приклад використання мікроконтролера ESP32 із датчиком DHT22. URL: https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/ (дата звернення: 07.06.2023).

21.Документація Wokwi. URL: https://docs.wokwi.com/?utm_source=wokwi (дата звернення: 07.06.2023).