Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Методи та засоби периферійних та хмарних обчислень в розумних містах

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Нестайко С. Т.
(підпис) (прізвище та ініціали)

Керівник _____ Мацюк Г. Р.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Липак Г. І.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І. О.
(підпис) (прізвище та ініціали)

Рецензент _____ Козак Р. О.
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Нестайку Станіславу Тарасовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи та засоби периферійних та хмарних обчислень в розумних містах

Керівник роботи Мацюк Галина Ростиславівна, канд. наук з соц. ком., доцент кафедри УІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 14 » травня 2026 року № 4/9-239

2. Термін подання студентом завершеної роботи 22 червня 2026р.

3. Вихідні дані до роботи Літературні та інтернет джерела інформації по технології розробки програмного комплексу для складання розкладу занять.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Постановка задачі та формування вимог до програмного комплексу. 1.2. Аналіз предметної області. 1.2. Формування вимог до програмного комплексу. 1.3. Пошук актантів та варіантів використання. 1.4. Вибір середовища розробки. 1.5. Обґрунтування використовуваних технологій. 1.6. Висновок до першого розділу. 2. Проєктування та методи розробки програмного комплексу. 2.1. Формування структури програмного комплексу. 2.2. Проєктевання архітектури системи обчислень. 2.3. Проєктування моделі взаємодії IoT-пристроїв, периферійних вузлів та хмарної платформи. 2.4. Проєктування та створення БД для розроблюваного комплексу. 2.5. Розробка алгоритму функціонування програмного комплексу. 2.6. Висновок до другого розділу. 3. Реалізація та тестування програмного комплексу. 3.1. Розробка інтерфейсу. 3.2. Розробка програмного комлпексу. 3.3. Тестування Розробленого програмного комплексу. 3.4. Перспективи модернізації. 3.5. Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. 4.1. Менеджмент безпеки 4.2. Методи боротьби з монотонністю праці. 4.3. Висновок до третього розділу. Висновки Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульна сторінка. 2. Мета кваліфікаційної роботи. 3. Актуальність обраної теми. 4. Аналіз предметної області. 5. Порівняльний аналіз периферійних та хмарних обчислень. 6. Огляд існуючих аналогів. 7. Використані технології. 8. Архітектура програмного комплексу. 9. Структура бази даних. 10. Користувацький інтерфейс. 11. Перспективи модернізації. 12. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О. Я. кандидат технічних наук, доцент кафедри МТ		

7. Дата видачі завдання 23 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	23.01.2026	*Виконано*
2.	Підбір джерел про технології розробки програмних	24.01.2026-26.01.2026	*Виконано*
	комплексів для моніторингу та обробки даних у		
	системах розумних міст		
3.	Опрацювання джерел по темі кваліфікаційної роботи	27.01.2026-31.01.2026	*Виконано*
4.	Виконання дослідження щодо розробки програмного	01.02.2026-07.02.2026	*Виконано*
	комплексу для моніторингу та обробки даних у		
	система розумних міст		
	Розроблення функціоналу програмного комплексу		
5.	Оформлення розділу «Постановка задачі та	08.02.2026-09.02.2026	*Виконано*
	формування вимог до програмного комплексу»		
6.	Оформлення розділу «Проєктування та методи	10.02.2026-12.02.2026	*Виконано*
	розробки програмного комплексу»		
7.	Оформлення розділу «Реалізація та тестування	14.02.2026-16.02.2026	*Виконано*
	програмного комплексу»		
8.	Виконання завдання до розділу «Безпека	05.06.2026-08.06.2026	*Виконано*
	життєдіяльності, основи охорони праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-13.06.2026	*Виконано*
11.	Перевірка на плагіат	19.06.2026	*Виконано*
12.	Попередній захист кваліфікаційної роботи		
13.	Захист кваліфікаційної роботи	24.06.2026	

Студент _____ Нестайко С. Т.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мацюк Г. Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Методи та засоби периферійних та хмарних обчислень в розумних містах // Кваліфікаційна робота освітнього рівня «Бакалавр» // Нестайко Станіслав Тарасович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. 59, рис. – 9, додат. – 2, бібліогр. – 35.

Ключові слова: розумне місто, периферійні обчислення, хмарні обчислення, IoT, програмний комплекс, моніторинг, обробка даних, інформаційна система

Кваліфікаційна робота присвячена дослідженню методів та засобів периферійних і хмарних обчислень у розумних містах та розробці програмного комплексу для збору, обробки, аналізу і візуалізації інформації, що надходить від периферійних пристроїв міської інфраструктури.

У першому розділі кваліфікаційної роботи розглянуто постановку задачі та сформовано вимоги до програмного комплексу. Проведено аналіз предметної області, досліджено особливості функціонування систем розумних міст, визначено основних користувачів і сценарії використання системи, а також обґрунтовано вибір сучасних технологій та програмних засобів для реалізації програмного комплексу.

У другому розділі виконано проєктування програмного комплексу. Сформовано його структуру, розроблено архітектуру системи та модель взаємодії основних компонентів, спроєктовано структуру бази даних і алгоритм функціонування програмного забезпечення. Запропоновані рішення забезпечують ефективний розподіл обчислювального навантаження між периферійними вузлами та хмарною інфраструктурою.

У третьому розділі кваліфікаційної роботи реалізовано програмний комплекс, розроблено користувацький інтерфейс та функції для обробки даних, виконано тестування основних модулів системи й розглянуто перспективи її подальшої модернізації. Результати тестування підтвердили стабільність роботи програмного комплексу та його відповідність поставленим вимогам.

У четвертому розділі досліджено питання безпеки життєдіяльності та охорони праці під час роботи з комп'ютерною технікою. Проаналізовано основні аспекти охорони праці та менеджменту безпеки, також було досліджено методи боротьби з монотонністю праці.

ANNOTATION

Methods and Means of Edge and Cloud Computing in Smart Cities // Bachelor's Qualification Thesis // Stanislav Tarasovych Nestaiko // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group CH-42 // Ternopil, 2026 // 59 pages, 9 figures, 2 appendices, 35 references.

Keywords: Smart City, edge computing, cloud computing, IoT, software system, monitoring, data processing, information system.

The qualification thesis is devoted to the study of methods and means of edge and cloud computing in smart cities and the development of a software system for collecting, processing, analyzing, and visualizing data received from edge devices of urban infrastructure.

The first chapter of the qualification thesis describes the problem statement and defines the requirements for the software system. It includes an analysis of the subject area, investigates the features of Smart City systems, identifies the main users and use cases, and substantiates the choice of modern technologies and software tools for implementing the proposed solution.

The second chapter focuses on the design of the software system. The overall structure of the system is developed, its architecture and interaction model between the main components are designed, the database structure is created, and the algorithm of the software operation is proposed. The presented solutions provide efficient distribution of computational load between edge nodes and cloud infrastructure.

The third chapter presents the implementation of the software system, including the development of the user interface and data processing functions, testing of the main functional modules, and consideration of further modernization possibilities. The testing results confirm the stability of the software system and its compliance with the specified requirements.

The fourth chapter addresses occupational safety and health issues related to working with computer equipment. The main aspects of occupational health and safety management were analyzed, and methods of combating labor monotony were also investigated.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – програмний інтерфейс застосунку, що забезпечує взаємодію між різними програмними компонентами та інформаційними системами.

Cloud Computing (англ. Cloud Computing) – модель надання обчислювальних ресурсів, програмного забезпечення та сховищ даних через мережу Інтернет.

CSS (англ. Cascading Style Sheets) – мова стилів, що використовується для опису зовнішнього вигляду вебсторінок.

Edge Computing (англ. Edge Computing) – технологія периферійних обчислень, за якої обробка інформації виконується безпосередньо поблизу джерела її створення.

HTTP (англ. HyperText Transfer Protocol) – протокол передачі гіпертексту, який використовується для обміну інформацією між клієнтом і сервером.

IoT (англ. Internet of Things) – концепція мережі взаємопов'язаних фізичних пристроїв, обладнаних датчиками та засобами передачі даних через мережу Інтернет.

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними, що використовується для передачі інформації між клієнтською та серверною частинами програмного забезпечення.

Node.js – програмна платформа для виконання JavaScript-коду на серверній стороні, побудована на рушії V8.

REST API (англ. Representational State Transfer Application Programming Interface) – архітектурний стиль взаємодії програмних компонентів через HTTP-запити.

Розумне місто (англ. Smart City) – концепція міста, що передбачає використання сучасних інформаційно-комунікаційних технологій для ефективного управління міською інфраструктурою та підвищення якості життя населення.

ЗМІСТ

ВСТУП 11
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОГО КОМПЛЕКСУ 13
1.1 Аналіз предметної області 13
1.2 Формування вимог до програмного комплексу 15
1.3 Аналіз існуючих методів периферійних та хмарних обчислень 17
1.4 Вибір середовища та засобів розробки 18
1.5 Обґрунтування використовуваних технологій 20
1.6 Висновок до першого розділу 22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА МЕТОДИ РОЗРОБКИ ПРОГРАМНОГО КОМПЛЕКСУ 23
2.1 Формування структури програмного комплексу 23
2.2 Проєктування архітектури системи периферійних та хмарних обчислень 24
2.3 Проєктування моделі взаємодії IoT-пристроїв, периферійних вузлів та хмарної платформи 26
2.4 Проєктування бази даних та інформаційної моделі системи 28
2.5 Розробка алгоритму функціонування програмного комплексу 31
2.6 Висновок до другого розділу 32
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ 34
3.1 Проєктування інтерфейсу користувача 34
3.2 Розробка програмного комплексу 35
3.2.1 Розробка інтерфейсу 37
3.2.2 Розробка функцій для обробки даних 39
3.3 Тестування розробленого програмного комплексу 42
3.4 Перспективи модернізації 46
3.5 Висновок до третього розділу 47

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 49
4.1 Менеджмент безпеки 49
4.2 Методи боротьби з монотонністю праці 52
4.3 Висновок до четвертого розділу 54
ВИСНОВКИ 55
ПЕРЕЛІК ДЖЕРЕЛ 57
ДОДАТКИ

ВСТУП

Актуальність теми. Стрімкий розвиток інформаційних технологій та концепції розумного міста зумовлює активне впровадження сучасних методів обробки даних у міській інфраструктурі. Зростання кількості підключених IoT-пристроїв, сенсорних мереж та інтелектуальних систем керування вимагає використання ефективних засобів збирання, аналізу та зберігання великих обсягів інформації. Саме тому все більшого поширення набувають технології периферійних (Edge Computing) та хмарних (Cloud Computing) обчислень, які дозволяють забезпечити швидку обробку даних і зменшити навантаження на центральні сервери

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є дослідження методів і засобів периферійних та хмарних обчислень у розумних містах та розробка програмного комплексу для збору, обробки, аналізу і візуалізації даних, що надходять від периферійних пристроїв.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати предметну область та сучасні підходи до побудови систем розумних міст;

– дослідити методи периферійних та хмарних обчислень і особливості їх застосування;

– сформувати функціональні та нефункціональні вимоги до програмного комплексу;

– спроєктувати архітектуру системи та модель взаємодії її компонентів;

– спроєктувати структуру бази даних для зберігання інформації;

– розробити програмний комплекс для моніторингу та обробки даних;

– реалізувати користувацький інтерфейс і функції аналізу інформації;

– провести тестування розробленого програмного комплексу та оцінити його ефективність

Практичне значення одержаних результатів. Розроблений у процесі виконання кваліфікаційної роботи програмний комплекс може бути використаний як основа для створення інформаційних систем підтримки розумних міст. Його використання дозволяє автоматизувати процес збору та обробки інформації від периферійних пристроїв, здійснювати централізований моніторинг міської інфраструктури, оперативно виявляти критичні події та формувати аналітичні звіти.

Практичне застосування запропонованого рішення сприятиме підвищенню ефективності роботи міських служб, скороченню часу реагування на аварійні ситуації, оптимізації використання обчислювальних ресурсів та покращенню якості управління інформаційними процесами в умовах функціонування сучасного розумного міста.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОГО КОМПЛЕКСУ

1.1 Аналіз предметної області

Стрімкий розвиток інформаційних технологій та концепції розумного міста (Smart City**)** зумовлює необхідність використання сучасних методів обробки, зберігання та аналізу великих обсягів даних, що надходять від численних пристроїв Інтернету речей (IoT). До таких пристроїв належать датчики температури, вологості, якості повітря, камери відеоспостереження, системи моніторингу дорожнього руху, інтелектуальні лічильники та інші елементи міської інфраструктури. Постійне зростання кількості таких пристроїв призводить до значного збільшення потоків даних, які потребують швидкої обробки та передачі.

Традиційна модель централізованих хмарних обчислень не завжди забезпечує необхідний рівень швидкодії, оскільки всі дані передаються до віддалених центрів обробки даних. Це збільшує затримку передачі інформації, підвищує навантаження на мережу та може негативно впливати на роботу систем, що потребують прийняття рішень у режимі реального часу.

Для вирішення цієї проблеми активно застосовується концепція периферійних (Edge Computing) обчислень, яка передбачає обробку інформації безпосередньо поблизу джерела її виникнення. Завдяки цьому значно скорочується час реагування системи, зменшується обсяг даних, що передаються до хмарної інфраструктури, а також підвищується надійність роботи сервісів.

Хмарні обчислення (Cloud Computing), у свою чергу, забезпечують централізоване зберігання інформації, масштабованість програмних ресурсів та можливість виконання складних аналітичних обчислень. Поєднання периферійних та хмарних технологій створює ефективну багаторівневу архітектуру, яка дозволяє оптимізувати використання обчислювальних ресурсів

та забезпечити стабільне функціонування інформаційних систем розумного міста.

Сучасні розумні міста використовують периферійні та хмарні обчислення у багатьох сферах діяльності. До них належать інтелектуальні транспортні системи, автоматизоване керування дорожнім рухом, моніторинг стану навколишнього середовища, системи відеоспостереження, управління енергоспоживанням, контроль роботи комунальних служб та системи громадської безпеки. Використання цих технологій сприяє підвищенню ефективності управління міською інфраструктурою та покращенню якості життя населення.

Однією з головних переваг периферійних обчислень є можливість локальної обробки критично важливих даних без необхідності постійного звернення до хмарних серверів. Це особливо важливо для систем екстреного реагування, управління транспортними потоками та автоматизованого керування міськими об'єктами, де навіть незначна затримка може призвести до негативних наслідків.

Водночас хмарна інфраструктура забезпечує довготривале зберігання інформації, виконання аналітичної обробки великих масивів даних, побудову прогнозних моделей та використання алгоритмів машинного навчання для підтримки прийняття управлінських рішень. Саме комплексне використання Edge Computing та Cloud Computing дозволяє створити сучасну, масштабовану та високопродуктивну інформаційну систему.

Предметом дослідження є методи та засоби периферійних і хмарних обчислень, що застосовуються для обробки інформації в інтелектуальних системах розумних міст.

Об'єктом дослідження виступають інформаційні процеси збору, передачі, обробки та зберігання даних у системах розумних міст.

Актуальність дослідження обумовлена постійним розвитком технологій Інтернету речей, збільшенням кількості підключених пристроїв та необхідністю створення високоефективних інформаційних систем, здатних забезпечувати

швидку обробку великих потоків даних. Використання периферійних та хмарних обчислень дозволяє значно підвищити продуктивність цифрової інфраструктури розумного міста, знизити навантаження на мережеві канали та забезпечити оперативне прийняття управлінських рішень.

Отже, дослідження методів і засобів периферійних та хмарних обчислень є важливим напрямом розвитку сучасних інформаційних технологій та має вагоме практичне значення для побудови ефективних систем управління розумними містами.

1.2 Формування вимог до програмного комплексу

Одним із найважливіших етапів розробки інформаційної системи для периферійних та хмарних обчислень у розумних містах є формування вимог до програмного комплексу. Саме на цьому етапі визначаються функціональні можливості системи, її архітектура, вимоги до продуктивності, безпеки та масштабованості. Правильно сформовані вимоги забезпечують створення ефективного програмного рішення, здатного працювати в умовах великої кількості підключених пристроїв та безперервного потоку даних.

Програмний комплекс повинен забезпечувати збір інформації від периферійних пристроїв, її попередню обробку на рівні Edge Computing та передачу до хмарної інфраструктури для подальшого аналізу і зберігання. Такий підхід дозволяє зменшити навантаження на мережеві канали, скоротити затримку передачі даних та підвищити швидкість реагування системи.

До основних функціональних вимог програмного комплексу належать:

— отримання даних від IoT-пристроїв у режимі реального часу;
— обробка та фільтрація інформації на периферійних вузлах;
— передача необхідних даних до хмарного сервера;
— централізоване зберігання інформації;
— моніторинг стану пристроїв та візуалізація показників;
— формування повідомлень про аварійні або критичні ситуації;

— підтримка віддаленого адміністрування системи.

Окрім функціональних характеристик, програмний комплекс повинен відповідати низці нефункціональних вимог. Насамперед система має забезпечувати високу продуктивність та можливість одночасної роботи з великою кількістю периферійних пристроїв. Важливою вимогою є масштабованість, яка дозволяє легко додавати нові модулі та вузли без зміни загальної архітектури.

Особливу увагу необхідно приділити питанням інформаційної безпеки. Передача даних між периферійними вузлами та хмарною платформою повинна здійснюватися із використанням сучасних методів шифрування та механізмів автентифікації користувачів. Це забезпечує захист інформації від несанкціонованого доступу та підвищує надійність роботи системи.

Програмний комплекс повинен мати інтуїтивно зрозумілий вебінтерфейс, який надаватиме можливість оператору переглядати поточний стан міської інфраструктури, аналізувати статистичні дані та оперативно реагувати на виникнення надзвичайних ситуацій. Інтерфейс повинен бути адаптований для роботи на різних пристроях та підтримувати сучасні вебтехнології.

Для реалізації системи доцільно використовувати клієнт-серверну архітектуру, де периферійні пристрої виконують первинну обробку інформації, серверна частина забезпечує взаємодію з базою даних і хмарними сервісами, а вебклієнт реалізує функції моніторингу та керування. Така структура дозволяє ефективно розподілити обчислювальні ресурси між Edge та Cloud середовищами.

Отже, формування вимог до програмного комплексу є основою його подальшого проєктування та реалізації. Визначення функціональних і нефункціональних характеристик системи забезпечує створення сучасного програмного рішення для підтримки периферійних та хмарних обчислень у розумних містах, здатного працювати в умовах високого навантаження та постійного збільшення кількості підключених пристроїв.

1.3 Аналіз існуючих методів периферійних та хмарних обчислень

Сучасний розвиток концепції розумних міст безпосередньо пов'язаний із використанням периферійних та хмарних обчислень. Ці технології дозволяють ефективно обробляти великі обсяги інформації, що надходять від датчиків, камер відеоспостереження, транспортних систем та інших елементів міської інфраструктури. Завдяки їх поєднанню забезпечується безперервний збір, аналіз та передача даних у режимі реального часу.

Традиційні хмарні обчислення передбачають централізовану обробку інформації у віддалених дата-центрах. Усі дані, отримані від пристроїв Інтернету речей, передаються через мережу до хмарної платформи, де виконуються їх обробка, аналіз та зберігання. Основними перевагами такого підходу є висока продуктивність серверів, масштабованість, централізоване адміністрування та можливість використання складних алгоритмів аналізу даних і машинного навчання.

Проте централізована модель має певні недоліки. Передача великих обсягів інформації до хмари збільшує навантаження на мережеві канали, що може призводити до затримок під час отримання результатів обробки. Це особливо критично для систем управління транспортом, відеоспостереження або екстреного реагування, де швидкість прийняття рішень має вирішальне значення.

Для усунення цих недоліків широко використовується технологія периферійних обчислень (Edge Computing). Її основною особливістю є виконання обчислень безпосередньо біля джерела виникнення даних. Частина інформації аналізується на локальних пристроях або периферійних серверах, після чого до хмарної платформи передаються лише результати обробки або узагальнені дані. Це дозволяє значно скоротити затримку передачі інформації та підвищити ефективність роботи всієї системи.

Окремим напрямом розвитку є технологія Fog Computing, яка займає проміжне місце між периферійними пристроями та хмарною інфраструктурою.

У такій моделі частина обчислень виконується на локальних мережевих вузлах, що дозволяє рівномірно розподілити навантаження між різними рівнями системи та забезпечити більш гнучку обробку інформації.

У сучасних розумних містах найчастіше використовується комбінований підхід, який поєднує переваги Edge Computing та Cloud Computing. Периферійні вузли виконують швидку локальну обробку даних і приймають оперативні рішення, тоді як хмарні сервери забезпечують довготривале зберігання інформації, аналітичну обробку, формування статистики та прогнозування розвитку подій.

Для передачі даних між компонентами системи використовуються сучасні мережеві протоколи, серед яких MQTT, HTTP, HTTPS, WebSocket та AMQP. Найбільш популярним у системах Інтернету речей є протокол MQTT, який характеризується невеликим обсягом службової інформації, високою швидкістю роботи та мінімальним навантаженням на мережу.

Важливою складовою сучасних систем є використання контейнеризації та мікросервісної архітектури. Застосування Docker, Kubernetes та інших інструментів дозволяє швидко розгортати окремі сервіси, масштабувати систему відповідно до навантаження та забезпечувати її безперервну роботу.

Таким чином, аналіз існуючих методів периферійних та хмарних обчислень показує, що найбільш ефективним рішенням для реалізації концепції розумного міста є поєднання локальної обробки інформації на периферійних вузлах із можливостями централізованих хмарних сервісів. Такий підхід забезпечує високу швидкодію, масштабованість, надійність та ефективне використання обчислювальних ресурсів, що робить його оптимальним для побудови сучасних інформаційних систем розумних міст.

1.4 Вибір середовища та засобів розробки

Одним із важливих етапів створення програмного комплексу для підтримки периферійних та хмарних обчислень у розумних містах є вибір

сучасних засобів і технологій розробки. Правильно підібране програмне забезпечення та інструменти дозволяють забезпечити високу продуктивність системи, її масштабованість, надійність та можливість подальшої модернізації.

Для розробки серверної частини програмного комплексу доцільно використовувати платформу Node.js**,** яка забезпечує асинхронну обробку великої кількості запитів та ефективну роботу з потоками даних у режимі реального часу. Завдяки використанню неблокуючої моделі введення-виведення Node.js широко застосовується при створенні IoT-систем та вебсервісів.

Для реалізації клієнтської частини програмного комплексу рекомендовано використовувати бібліотеку React**,** яка дозволяє створювати сучасні інтерактивні вебінтерфейси. Компонентний підхід спрощує підтримку програмного забезпечення та забезпечує швидке оновлення інформації без перезавантаження сторінки.

Як систему керування базами даних доцільно використовувати MongoDB**.** Документоорієнтована структура цієї бази даних добре підходить для зберігання великої кількості різнорідної інформації, що надходить від периферійних пристроїв. MongoDB забезпечує високу швидкість роботи та можливість горизонтального масштабування.

Для організації взаємодії між IoT-пристроями та серверною частиною системи рекомендується використовувати протокол MQTT, який характеризується мінімальним навантаженням на мережу та підтримує швидку передачу повідомлень між клієнтами й сервером.

Середовищем розробки програмного комплексу може бути Visual Studio Code, яке підтримує велику кількість мов програмування, має інтегровані засоби налагодження, систему керування версіями Git та широкий набір розширень для роботи з Node.js, React і MongoDB.

Контроль версій програмного коду рекомендується здійснювати за допомогою системи Git, а для зберігання проєкту використовувати платформу GitHub, яка забезпечує спільну роботу над програмним забезпеченням та контроль історії змін.

Таким чином, обраний комплекс програмних засобів забезпечує створення сучасної інформаційної системи для підтримки периферійних та хмарних обчислень у розумних містах. Використання Node.js, React, MongoDB, MQTT та Git дозволяє реалізувати масштабований, надійний і високопродуктивний програмний комплекс, здатний ефективно працювати з великими потоками даних у режимі реального часу.

1.5 Обґрунтування використовуваних технологій

Розробка сучасних інформаційних систем для підтримки концепції розумного міста потребує використання програмних технологій, які забезпечують високу продуктивність, масштабованість, надійність та можливість обробки великих потоків інформації в режимі реального часу. Саме тому під час створення програмного комплексу було обрано сучасний стек технологій, який дозволяє ефективно реалізувати взаємодію між периферійними пристроями, серверною частиною та хмарною інфраструктурою.

Для реалізації серверної частини системи обрано платформу Node.js, яка побудована на високопродуктивному JavaScript-рушії V8. Її головною перевагою є асинхронна модель виконання операцій, що дозволяє одночасно обробляти велику кількість запитів без суттєвого навантаження на сервер. Це особливо важливо для систем розумних міст, де інформація безперервно надходить від великої кількості датчиків та IoT-пристроїв.

Для створення користувацького інтерфейсу використано бібліотеку React, яка забезпечує компонентний підхід до побудови вебзастосунків. Завдяки використанню Virtual DOM оновлення інтерфейсу відбувається швидко та без повного перезавантаження сторінки, що позитивно впливає на швидкодію системи та зручність роботи користувача.

Як систему керування базами даних обрано MongoDB, яка належить до класу NoSQL-баз даних. Її документоорієнтована модель дозволяє гнучко зберігати інформацію різної структури, що є важливою перевагою при роботі з

даними, отриманими від різних периферійних пристроїв. Крім того, MongoDB підтримує горизонтальне масштабування та забезпечує високу швидкість виконання операцій.

Для передачі інформації між IoT-пристроями та серверною частиною системи використовується протокол MQTT, який спеціально розроблений для мереж з обмеженою пропускною здатністю. Протокол характеризується мінімальним обсягом службової інформації, високою швидкістю передачі повідомлень та низьким енергоспоживанням, що робить його оптимальним для застосування в системах Інтернету речей.

Для розгортання хмарної інфраструктури можуть використовуватися сучасні хмарні платформи, які забезпечують централізоване зберігання інформації, резервне копіювання даних, автоматичне масштабування сервісів та можливість віддаленого доступу до інформаційних ресурсів. Це дозволяє підтримувати стабільну роботу системи незалежно від кількості підключених користувачів та периферійних вузлів.

Застосування системи контролю версій Git забезпечує ефективне керування програмним кодом, можливість спільної роботи розробників та контроль усіх внесених змін. Використання репозиторіїв дозволяє швидко відновлювати попередні версії проєкту та організовувати безпечний процес розробки.

Обрані технології добре поєднуються між собою, підтримують сучасні стандарти веброзробки та дозволяють реалізувати багаторівневу архітектуру програмного комплексу. Вони забезпечують високу продуктивність, масштабованість і надійність системи, що є необхідними умовами для ефективного функціонування периферійних та хмарних обчислень у розумних містах.

Отже, використання сучасних програмних технологій і засобів розробки створює передумови для побудови ефективної інформаційної системи, здатної забезпечувати швидку обробку даних, підтримувати взаємодію між різними компонентами інфраструктури та сприяти розвитку концепції розумних міст.

1.6 Висновок до першого розділу

У першому розділі кваліфікаційної роботи було проведено аналіз предметної області, пов’язаної із застосуванням периферійних та хмарних обчислень у розумних містах. Встановлено, що сучасні системи розумних міст функціонують на основі великої кількості взаємопов’язаних пристроїв Інтернету речей, які генерують значні обсяги інформації та потребують ефективних методів її збору, передачі, обробки й зберігання.

У процесі дослідження було сформовано основні вимоги до програмного комплексу, визначено його функціональні та нефункціональні характеристики, а також окреслено ключові завдання, які повинна виконувати інформаційна система. Особливу увагу приділено необхідності забезпечення високої швидкодії, масштабованості, надійності та безпеки під час роботи з великими потоками даних.

Отримані результати дослідження є теоретичною основою для подальшого проєктування програмного комплексу, розробки його архітектури, моделювання взаємодії компонентів та реалізації основних функціональних можливостей, що будуть розглянуті у наступному розділі роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА МЕТОДИ РОЗРОБКИ ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Формування структури програмного комплексу

Проєктування структури програмного комплексу є одним із ключових етапів розробки системи периферійних та хмарних обчислень для розумного міста. Від правильно побудованої архітектури залежить ефективність роботи програмного забезпечення, можливість його масштабування, швидкість обробки інформації та надійність функціонування в умовах великої кількості підключених пристроїв.

Основною метою розроблюваного програмного комплексу є забезпечення збору, передачі, обробки та аналізу інформації, що надходить від периферійних пристроїв, а також її централізованого зберігання у хмарній інфраструктурі. Для цього система реалізується за багаторівневою архітектурою, що складається з рівня збору даних, периферійного рівня обробки, хмарного сервера та клієнтського застосунку.

Першим рівнем структури є IoT-пристрої, які здійснюють вимірювання різних параметрів міської інфраструктури. До них можуть належати датчики температури, вологості, рівня освітленості, якості повітря, камери відеоспостереження, інтелектуальні лічильники та інші засоби моніторингу. Саме ці пристрої є основними джерелами інформації для всієї системи.

Наступним елементом структури виступає периферійний (Edge) вузол, який виконує первинну обробку отриманих даних. На цьому рівні здійснюється фільтрація інформації, перевірка її коректності, усунення дублювання та формування повідомлень про критичні події. Використання Edge Computing дозволяє суттєво скоротити навантаження на мережу та зменшити час реагування системи [1], [2].

Оброблена інформація передається до хмарної платформи, яка забезпечує довготривале зберігання даних, виконання аналітичних обчислень, побудову

статистичних звітів та прогнозування. Хмарна інфраструктура також відповідає за резервне копіювання інформації та підтримку масштабованості програмного комплексу.

Для взаємодії користувачів із системою передбачено вебінтерфейс, який надає можливість переглядати поточний стан міської інфраструктури, аналізувати отримані показники, будувати графіки та здійснювати адміністрування системи. Інтерфейс повинен бути інтуїтивно зрозумілим, адаптивним та забезпечувати швидкий доступ до необхідної інформації.

Між усіма компонентами програмного комплексу організовується постійний обмін даними за допомогою сучасних мережевих протоколів. Такий підхід забезпечує безперервне функціонування системи, своєчасне оновлення інформації та підтримку роботи великої кількості периферійних пристроїв.

Структура програмного комплексу побудована за модульним принципом, що дозволяє легко додавати нові функціональні компоненти без внесення суттєвих змін до вже існуючих модулів. Це забезпечує гнучкість системи та можливість її подальшого розвитку відповідно до нових вимог концепції розумних міст [3].

Отже, сформована структура програмного комплексу забезпечує ефективну взаємодію між периферійними пристроями, локальними вузлами обробки, хмарною платформою та користувачами системи. Запропонована архітектура дозволяє оптимально розподілити обчислювальне навантаження, підвищити продуктивність роботи та створити надійну основу для подальшої реалізації інформаційної системи розумного міста.

2.2 Проєктування архітектури системи периферійних та хмарних обчислень

Архітектура програмного комплексу є основою ефективного функціонування системи та визначає спосіб взаємодії всіх її компонентів. Для реалізації системи периферійних та хмарних обчислень у розумному місті

доцільно використовувати багаторівневу архітектуру, яка забезпечує розподіл обчислювального навантаження між периферійними вузлами та хмарною платформою.

Запропонована архітектура складається з чотирьох основних рівнів: рівня збору даних, рівня периферійних обчислень (Edge Computing), рівня хмарних обчислень (Cloud Computing) та рівня користувацьких застосунків. Кожен із цих рівнів виконує власні функції та взаємодіє з іншими компонентами системи. Схематичне зображення архітектури програмного комплексу наведено у Додатку А, де відображено взаємозв'язок між основними елементами системи та напрямки передачі інформаційних потоків [4].

Перший рівень представлений IoT-пристроями, які здійснюють постійний моніторинг різних параметрів міської інфраструктури. До них належать датчики температури, вологості, якості повітря, транспортні сенсори, системи відеоспостереження, інтелектуальні лічильники та інші пристрої збору інформації. Вони генерують значний обсяг даних, який передається для подальшої обробки.

Другий рівень складається з периферійних серверів або шлюзів Edge Computing. На цьому рівні виконується попередня обробка інформації, її фільтрація, перевірка достовірності, агрегування та аналіз критичних подій. Завдяки локальній обробці даних система значно скорочує затримки передачі інформації та зменшує навантаження на мережеву інфраструктуру.

Третій рівень представлений хмарною платформою, яка забезпечує централізоване зберігання інформації, резервне копіювання, виконання складних аналітичних операцій, побудову статистичних моделей та використання алгоритмів прогнозування. Саме хмарна інфраструктура дозволяє масштабувати систему відповідно до кількості підключених пристроїв та обсягу інформації.

Четвертий рівень складається з користувацьких застосунків, через які оператори, адміністратори та інші користувачі взаємодіють із системою. За допомогою вебінтерфейсу вони можуть переглядати поточні показники,

аналізувати статистичні дані, отримувати повідомлення про аварійні ситуації та здійснювати керування окремими елементами міської інфраструктури.

Взаємодія між усіма компонентами архітектури здійснюється через мережеві протоколи передачі даних, що забезпечують швидкий і безпечний обмін інформацією. Така організація дозволяє досягти високої продуктивності системи, мінімізувати затримки та забезпечити безперервне функціонування сервісів розумного міста.

Запропонована архітектура є модульною, що дозволяє інтегрувати нові пристрої, сервіси та програмні модулі без необхідності повної перебудови системи. Це забезпечує її адаптивність до майбутніх технологічних змін та розширення функціональних можливостей [5], [6], [7].

Отже, проєктування архітектури системи периферійних та хмарних обчислень дозволяє створити надійну та масштабовану платформу для підтримки інформаційних процесів у розумному місті. Запропонована структура забезпечує ефективний розподіл обчислювальних ресурсів між периферійними вузлами та хмарною інфраструктурою, що підвищує швидкодію, надійність та ефективність роботи всієї системи.

2.3 Проєктування моделі взаємодії IoT-пристроїв, периферійних вузлів та хмарної платформи

Одним із найважливіших етапів створення програмного комплексу для підтримки периферійних та хмарних обчислень є проєктування моделі взаємодії між усіма компонентами системи. Від правильно організованого обміну інформацією залежить швидкість обробки даних, продуктивність системи та ефективність прийняття управлінських рішень у розумному місті.

Запропонована модель передбачає багаторівневу передачу інформації, у якій кожен компонент виконує власні функції. Основним джерелом даних виступають IoT-пристрої, що здійснюють постійний моніторинг навколишнього середовища та стану міської інфраструктури. До них належать датчики

температури, вологості, якості повітря, освітленості, транспортні сенсори, камери відеоспостереження та інші інтелектуальні пристрої.

Зібрані дані надходять до периферійних вузлів (Edge Computing), де виконується їх попередня обробка. На цьому етапі здійснюється перевірка достовірності інформації, усунення дубльованих записів, фільтрація неактуальних даних та формування повідомлень про критичні ситуації. Такий підхід дозволяє значно скоротити обсяг інформації, яка передається до хмарної інфраструктури, та зменшити навантаження на канали зв'язку.

Після локальної обробки результати передаються до хмарної платформи, де здійснюється їх довготривале зберігання, накопичення статистики, аналіз великих масивів даних та формування прогнозних моделей. Хмарна інфраструктура також забезпечує централізоване керування всіма периферійними вузлами та підтримує синхронізацію інформації між різними компонентами системи.

Важливим елементом моделі є користувацький рівень, який представлений вебзастосунком або мобільним інтерфейсом. Через нього оператори можуть переглядати поточний стан системи, аналізувати показники, отримувати повідомлення про надзвичайні ситуації та формувати статистичні звіти. Завдяки цьому забезпечується оперативне реагування на зміни у функціонуванні міської інфраструктури.

Для передачі інформації між компонентами системи можуть використовуватися сучасні мережеві протоколи MQTT, HTTP або HTTPS, які забезпечують швидкий, надійний та захищений обмін даними. Це дозволяє підтримувати стабільну роботу програмного комплексу навіть за умов великої кількості одночасно підключених пристроїв [8], [9].

Запропонована модель взаємодії реалізує принцип розподілених обчислень, за якого ресурсоємні операції виконуються у хмарному середовищі, а обробка критично важливих даних здійснюється безпосередньо на периферійних вузлах. Такий підхід дозволяє підвищити продуктивність системи, скоротити час відгуку та забезпечити безперервність її роботи.

Схема взаємодії IoT-пристроїв, периферійних вузлів, хмарної платформи та користувацького інтерфейсу наведена на рисунку 2.1. Вона демонструє основні інформаційні потоки між компонентами програмного комплексу та послідовність обробки даних від моменту їх збору до відображення результатів користувачу.

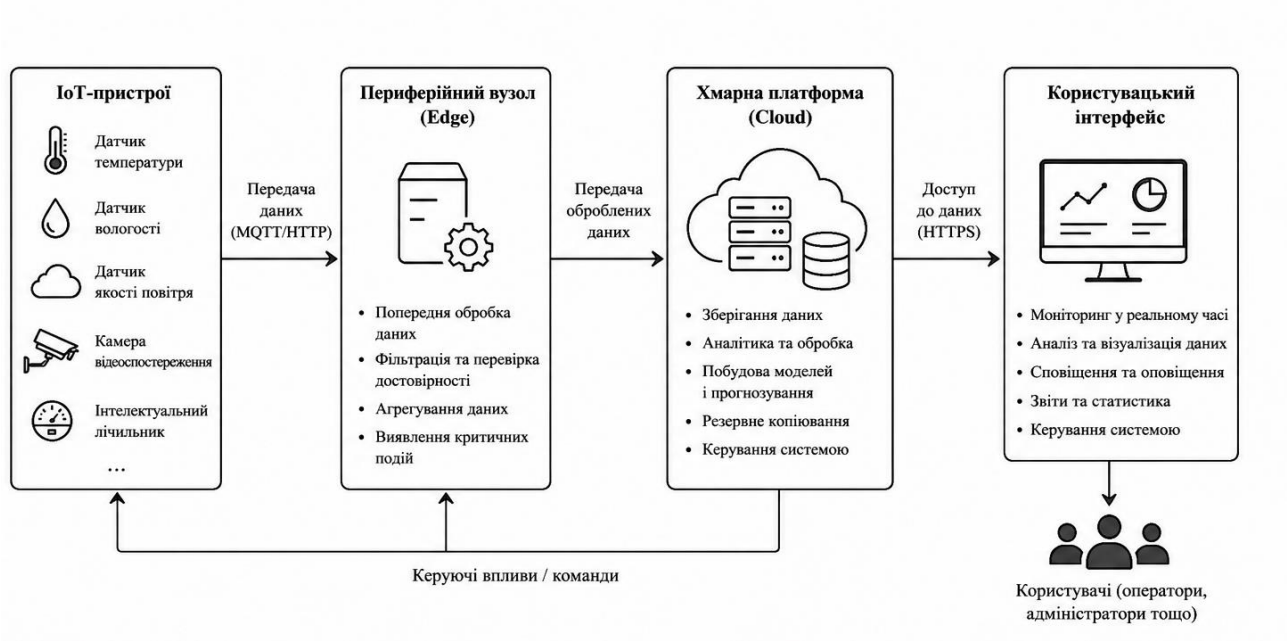


Рисунок 2.1 – Модель взаємодії

Отже, розроблена модель взаємодії забезпечує ефективний обмін інформацією між усіма елементами системи, оптимізує використання обчислювальних ресурсів та створює основу для побудови сучасної інформаційної системи підтримки

2.4 Проєктування бази даних та інформаційної моделі системи

Одним із найважливіших етапів розробки програмного комплексу для підтримки периферійних та хмарних обчислень є проєктування бази даних. Саме база даних забезпечує централізоване зберігання інформації, її швидкий пошук, оновлення та подальший аналіз. Правильно спроєктована інформаційна модель сприяє ефективній роботі всієї системи та забезпечує можливість її масштабування відповідно до потреб розумного міста.

Інформаційна система повинна накопичувати дані, що надходять від великої кількості периферійних пристроїв, а також зберігати інформацію про користувачів, події, повідомлення та результати обробки даних. Тому структура бази даних має бути гнучкою, надійною та оптимізованою для роботи з великими потоками інформації [10].

Основними сутностями інформаційної моделі є користувачі, IoT-пристрої, отримані показники, події системита сповіщення. Кожна із цих сутностей містить набір атрибутів, необхідних для забезпечення роботи програмного комплексу.

Сутність «Користувач» містить інформацію про осіб, які мають доступ до системи. До основних полів належать унікальний ідентифікатор, прізвище та ім'я, електронна адреса, пароль, роль користувача та дата реєстрації. Це дозволяє реалізувати систему автентифікації та розмежування прав доступу.

Сутність «IoT-пристрій» призначена для зберігання інформації про підключені датчики та інші периферійні пристрої. Для кожного пристрою зберігаються його ідентифікатор, назва, тип, місце розташування, поточний стан та дата підключення до системи.

Сутність «Показники» містить результати вимірювань, отримані від периферійних пристроїв. До неї входять значення параметра, одиниця вимірювання, час отримання інформації та посилання на відповідний пристрій. Таке структурування дозволяє швидко виконувати аналіз історичних даних та будувати статистичні звіти.

Сутність «Події» використовується для фіксації аварійних або нестандартних ситуацій, що виникають у процесі роботи системи. Для кожної події зберігаються її тип, опис, дата виникнення, рівень критичності та статус опрацювання [11].

Для оперативного інформування операторів використовується сутність «Сповіщення», яка містить інформацію про сформовані системою повідомлення, час їх створення, текст повідомлення та стан його перегляду користувачем.

Між сутностями бази даних існують логічні зв'язки, що забезпечують цілісність інформації та можливість швидкого отримання необхідних даних. Один користувач може переглядати велику кількість подій і сповіщень, а кожен IoT-пристрій може формувати необмежену кількість показників та повідомлень протягом свого функціонування.

Схему інформаційної моделі та структуру бази даних програмного комплексу наведено на рисунку 2.2, де відображено основні сутності, їх атрибути та взаємозв'язки між ними.

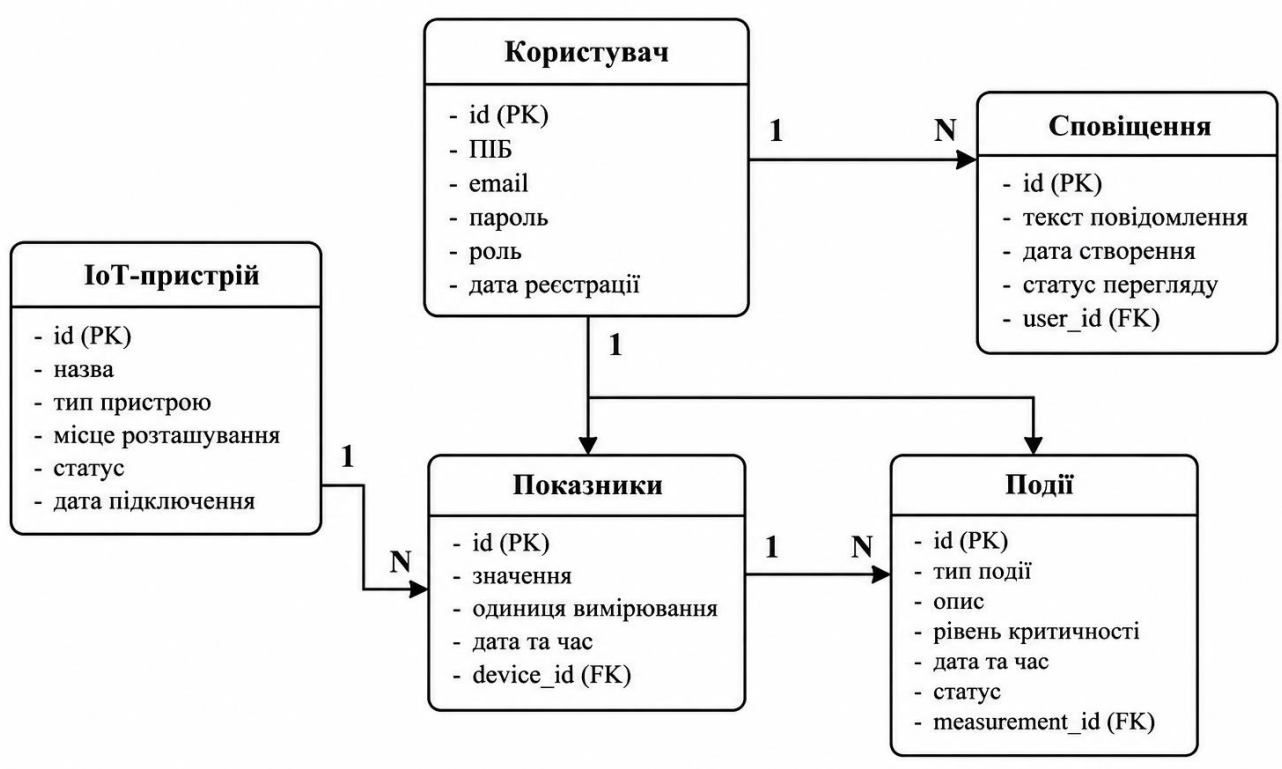


Рисунок 2.2 – Структура бази даних

Отже, спроєктована база даних забезпечує ефективне зберігання, обробку та аналіз інформації, що надходить від периферійних пристроїв. Запропонована інформаційна модель є масштабованою, відповідає вимогам сучасних інформаційних систем та створює основу для подальшої реалізації програмного комплексу підтримки периферійних та хмарних обчислень у розумних містах.

2.5 Розробка алгоритму функціонування програмного комплексу

Алгоритм функціонування програмного комплексу визначає послідовність виконання операцій, необхідних для збору, передачі, обробки, аналізу та відображення інформації в системі периферійних та хмарних обчислень розумного міста. Його правильна побудова забезпечує безперервність роботи системи, мінімізує час обробки інформації та дозволяє своєчасно реагувати на зміни параметрів міського середовища.

Робота програмного комплексу починається з отримання інформації від периферійних пристроїв, розташованих у різних частинах міста. Датчики здійснюють вимірювання заданих параметрів через визначені проміжки часу та передають отримані значення на периферійний сервер.

Після отримання інформації периферійний вузол виконує її попередню обробку. На цьому етапі здійснюється перевірка правильності отриманих даних, видалення дубльованих записів, аналіз допустимих меж показників та фільтрація некоректної інформації. Якщо значення виходять за встановлені межі, система автоматично формує повідомлення про можливу аварійну ситуацію.

Оброблені дані передаються до хмарної платформи, де здійснюється їх накопичення у базі даних та подальший аналіз. Хмарний сервер виконує статистичну обробку інформації, формує аналітичні звіти та забезпечує можливість довготривалого зберігання історичних даних [12], [13].

Наступним етапом є обробка запитів користувачів. Після авторизації оператор отримує доступ до вебінтерфейсу системи, де може переглядати поточний стан усіх підключених пристроїв, аналізувати статистичні показники та отримувати інформацію про критичні події.

У випадку виникнення аварійної ситуації програмний комплекс автоматично створює запис у журналі подій та надсилає сповіщення відповідальному користувачу. Це дозволяє значно скоротити час реагування на несправності та забезпечити оперативне прийняття управлінських рішень.

Алгоритм роботи системи побудований таким чином, щоб забезпечити безперервний цикл збору, аналізу та передачі інформації. Після завершення одного циклу система автоматично переходить до наступного етапу моніторингу, що гарантує постійне оновлення інформації та підтримку актуального стану даних.

Важливою особливістю алгоритму є можливість масштабування. При збільшенні кількості периферійних пристроїв або розширенні функціональних можливостей програмного комплексу його логіка роботи не змінюється, а лише збільшується кількість одночасно виконуваних операцій.

Схему алгоритму функціонування програмного комплексу наведено у Додатку А, де графічно представлено основні етапи роботи системи — від отримання інформації з IoT-пристроїв до її аналізу та відображення користувачу.

Отже, розроблений алгоритм забезпечує послідовне виконання всіх операцій програмного комплексу, ефективний розподіл обчислювального навантаження між периферійними та хмарними ресурсами, а також високу швидкодію й надійність роботи системи в умовах функціонування розумного міста.

2.6 Висновок до другого розділу

У другому розділі кваліфікаційної роботи було розглянуто основні підходи до проєктування програмного комплексу для реалізації периферійних та хмарних обчислень у розумних містах. Проведено формування структури системи, визначено її основні компоненти та принципи взаємодії між ними.

У ході роботи було спроєктовано архітектуру програмного комплексу, яка передбачає використання багаторівневої моделі із застосуванням IoT-пристроїв, периферійних вузлів, хмарної платформи та користувацького інтерфейсу. Такий підхід дозволяє ефективно розподіляти обчислювальне навантаження, зменшувати затримки під час обробки інформації та забезпечувати стабільну роботу системи.

Запропоновані рішення відповідають сучасним вимогам до інформаційних систем класу розумних міст, забезпечують можливість інтеграції нових пристроїв та сервісів, а також створюють основу для подальшого розвитку й удосконалення програмного комплексу.

Отже, результати, отримані у другому розділі, підтверджують доцільність використання технологій периферійних та хмарних обчислень для побудови сучасних систем розумного міста та створюють теоретичну й практичну основу для їх подальшого впровадження.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача є одним із найважливіших етапів розробки програмного комплексу для реалізації периферійних та хмарних обчислень у розумних містах. Від зручності та зрозумілості інтерфейсу залежить ефективність роботи операторів системи, швидкість прийняття рішень та якість моніторингу міської інфраструктури.

Під час проєктування інтерфейсу було враховано сучасні принципи UX/UI-дизайну, які передбачають логічне розташування елементів керування, мінімальну кількість дій для виконання основних операцій та швидкий доступ до необхідної інформації. Інтерфейс побудований таким чином, щоб користувач міг легко орієнтуватися у функціональних можливостях системи без додаткового навчання.

Головна сторінка програмного комплексу містить інформаційну панель, на якій відображаються основні показники роботи системи, кількість підключених периферійних пристроїв, поточний стан серверної інфраструктури та повідомлення про критичні події. Це дозволяє оператору оперативно оцінювати стан усієї мережі та швидко реагувати на можливі відхилення.

Навігація між функціональними модулями здійснюється за допомогою бокового меню, яке забезпечує швидкий перехід до розділів моніторингу, керування пристроями, перегляду статистики, журналу подій та адміністрування користувачів. Така структура сприяє зручній організації роботи та мінімізує час пошуку необхідної інформації [14].

Одним із основних елементів інтерфейсу є модуль візуалізації даних, який забезпечує відображення інформації у вигляді таблиць, графіків та діаграм. Це дозволяє аналізувати зміну показників у часі, виявляти тенденції та приймати обґрунтовані управлінські рішення.

Особливу увагу приділено адаптивності інтерфейсу. Вебзастосунок коректно працює на різних типах пристроїв, включаючи персональні комп'ютери, планшети та смартфони, що забезпечує можливість віддаленого моніторингу системи та оперативного доступу до інформації незалежно від місця перебування користувача.

Для забезпечення безпеки доступу до функціональних можливостей програмного комплексу передбачено систему авторизації та розмежування прав користувачів. Залежно від ролі користувача система надає доступ лише до тих функцій, які необхідні для виконання його посадових обов'язків [15].

Отже, спроєктований інтерфейс користувача забезпечує зручну взаємодію із системою, швидкий доступ до інформації, ефективне керування периферійними та хмарними обчислювальними ресурсами і відповідає сучасним вимогам до програмних комплексів для розумних міст.

3.2 Розробка програмного комплексу

Розробка програмного комплексу для реалізації системи периферійних та хмарних обчислень у розумних містах здійснювалася відповідно до сформованих вимог та спроєктованої архітектури системи. Основною метою розробки було створення програмного продукту, який забезпечує збір, передачу, обробку, зберігання та візуалізацію інформації, що надходить від периферійних пристроїв міської інфраструктури.

Програмний комплекс реалізовано за клієнт-серверною архітектурою, що дозволяє розділити функції між окремими компонентами системи. Клієнтська частина відповідає за взаємодію користувача із системою та відображення інформації, серверна частина забезпечує обробку запитів і виконання бізнес-логіки, а база даних використовується для централізованого зберігання всієї інформації.

Під час розробки особливу увагу приділено модульному принципу побудови програмного забезпечення. Окремі модулі відповідають за управління

користувачами, периферійними пристроями, обробку показників датчиків, журнал подій, систему повідомлень та аналітичний модуль. Такий підхід забезпечує незалежність окремих компонентів, спрощує супровід системи та дозволяє без значних змін інтегрувати нові функціональні можливості.

Для реалізації взаємодії між клієнтською та серверною частинами використовується REST API, що забезпечує швидкий та безпечний обмін інформацією. Передача даних здійснюється у форматі JSON, який є універсальним і підтримується більшістю сучасних програмних платформ [16], [17], [18].

Однією з основних функцій програмного комплексу є отримання даних від периферійних пристроїв у режимі реального часу. Отримана інформація проходить попередню перевірку, після чого зберігається у базі даних та використовується для побудови статистичних звітів, графіків і формування повідомлень про критичні ситуації.

Для підвищення продуктивності програмного комплексу реалізовано механізм оптимізації запитів до бази даних та мінімізації обсягу інформації, що передається між компонентами системи. Це дозволяє зменшити час обробки запитів та забезпечити стабільну роботу системи навіть при великій кількості одночасно підключених пристроїв.

У процесі розробки також реалізовано систему авторизації користувачів та розмежування прав доступу. Це забезпечує захист інформації та надає різним категоріям користувачів доступ лише до тих функцій, які необхідні для виконання їхніх завдань.

Таким чином, розроблений програмний комплекс забезпечує ефективне функціонування системи периферійних та хмарних обчислень у розумних містах, підтримує обробку великих обсягів інформації, централізоване керування пристроями та створює основу для подальшого розвитку інформаційної інфраструктури розумного міста [19].

3.2.1 Розробка інтерфейсу

Розробка інтерфейсу користувача є одним із найважливіших етапів створення програмного комплексу для реалізації периферійних та хмарних обчислень у розумних містах. Головною метою цього етапу було створення зручного, інтуїтивно зрозумілого та функціонального інтерфейсу, який забезпечує ефективну взаємодію користувача із системою.

Інтерфейс програмного комплексу реалізовано у вигляді вебзастосунку, що надає можливість доступу до всіх функцій через стандартний веббраузер. Під час розробки особливу увагу приділено адаптивності сторінок, що забезпечує їх коректне відображення на персональних комп'ютерах, планшетах та мобільних пристроях.

Головна сторінка містить інформаційну панель із короткою статистикою роботи системи, кількістю активних периферійних пристроїв, повідомленнями про критичні події та основними показниками функціонування міської інфраструктури. Навігація між окремими розділами здійснюється за допомогою бокового меню, що забезпечує швидкий доступ до всіх функціональних модулів.

Для покращення сприйняття інформації реалізовано систему графічного відображення даних у вигляді таблиць, діаграм та графіків. Це дозволяє оператору швидко оцінювати поточний стан системи та приймати оперативні управлінські рішення.

Інтерфейс підтримує систему авторизації користувачів та розмежування прав доступу. Після успішного входу до системи користувач отримує доступ лише до тих функцій, які відповідають його ролі та рівню повноважень.

Макети основних сторінок вебзастосунку наведено на рисунках 3.2-3.4, де представлено структуру інформаційної панелі, сторінки моніторингу та модуля керування периферійними пристроями.

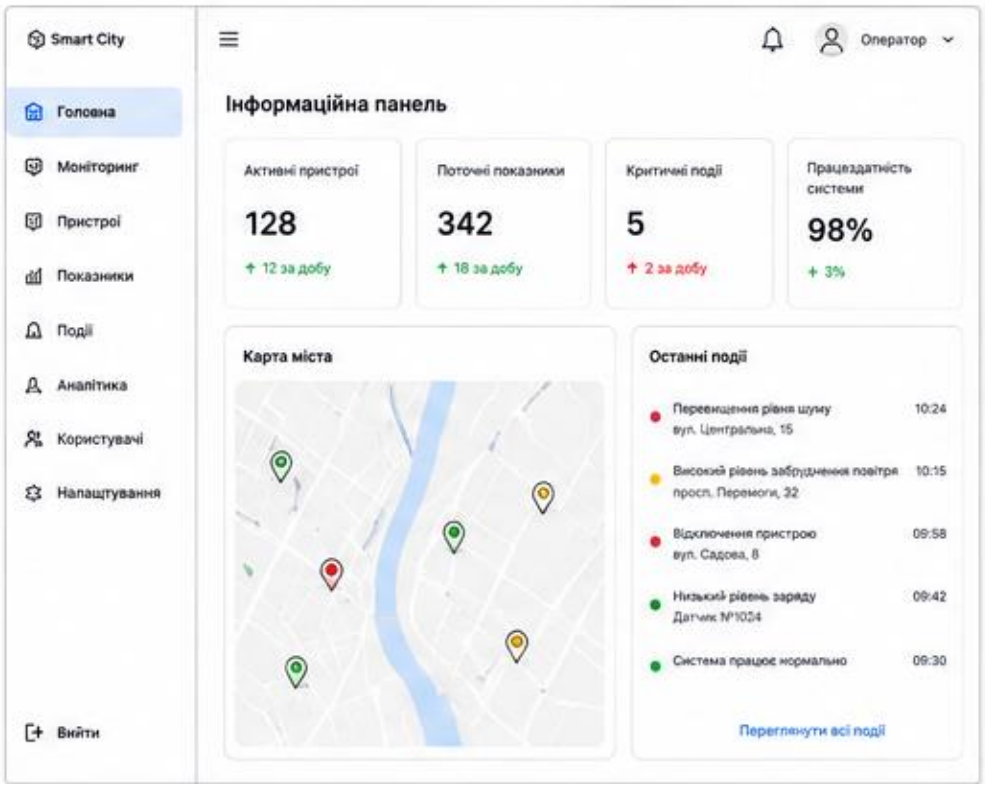


Рисунок 3.2 – Макет головної сторінки

На макеті головної сторінки зображено основні сторінки та інформація для користувача в зручному вигляді. Для кожного користувача ця сторінка буде частково відрізнятись виглядом через обмеженення функцій кожного користувача

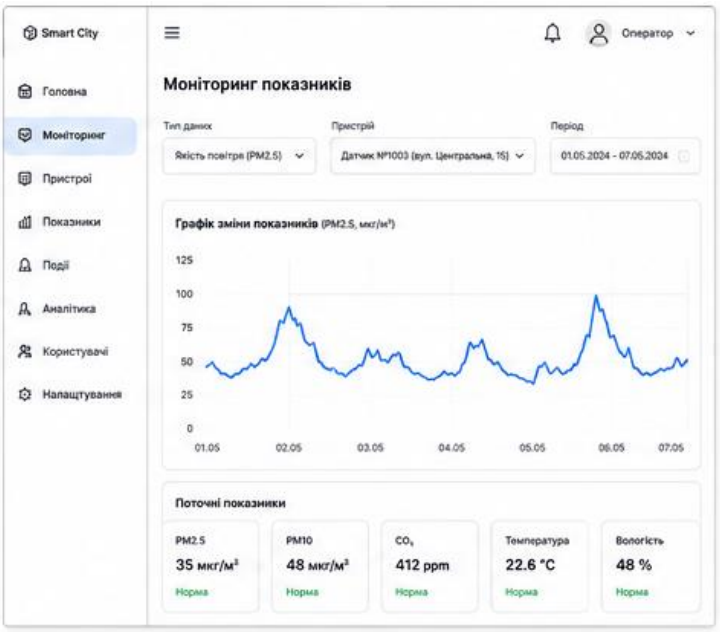


Рисунок 3.3 – Макет сторінки моніторингу

На рис 3.3 відображено макет сторінки моніторингу, де відображені основні функції, а саме моніторинг показників, графік змін та поточні показники. Всі показники змінюються в режимі реального часу, що допомагає користувачу вчасно помічати необхідні зміни.

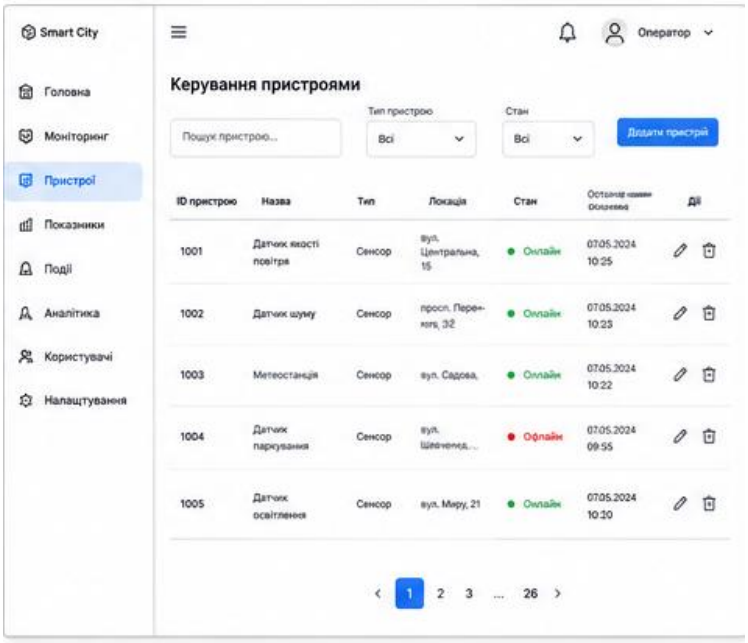


Рисунок 3.4 – Макет сторінки керування пристроями

Отже, розроблений інтерфейс користувача забезпечує зручну навігацію, швидкий доступ до необхідної інформації та відповідає сучасним вимогам до програмних комплексів для систем розумних міст.

3.2.2 Розробка функцій для обробки даних

Одним із найважливіших етапів створення програмного комплексу є реалізація функцій, що забезпечують приймання, обробку, аналіз та збереження інформації, отриманої від периферійних пристроїв. Саме програмні модулі обробки даних відповідають за ефективне функціонування всієї системи та взаємодію між її окремими компонентами [20], [21].

У процесі розробки було реалізовано набір функцій, які забезпечують отримання даних від периферійних вузлів, їх попередню перевірку, збереження

у базі даних, аналіз отриманих показників та передачу інформації до користувацького інтерфейсу.

Після надходження даних на сервер система автоматично виконує перевірку їх цілісності та коректності. У разі успішної валідації інформація передається до відповідного модуля обробки, де здійснюється її аналіз та підготовка до збереження.

Основна функція приймання даних відповідає за отримання інформації через API та формування внутрішньої структури даних програмного комплексу. Приклад реалізації цього програмного модуля наведено у лістингу 3.1.

Лістинг 3.1 – функція приймання даних від iot-пристроїв

```
app.post('/api/sensors/data', async (req, res) => {
    try {
        const sensorData = req.body;

        await SensorData.create({
            deviceId: sensorData.deviceId,
            temperature: sensorData.temperature,
            humidity: sensorData.humidity,
            airQuality: sensorData.airQuality,
            createdAt: new Date()
        });

        res.status(200).json({
            success: true,
            message: 'Дані успішно отримано'
        });
    } catch (error) {
        res.status(500).json({
            success: false,
            message: error.message
        });
    }
```

Наступним етапом є виконання функції збереження інформації до бази даних. Після перевірки отриманих параметрів система автоматично створює новий запис або оновлює існуючі дані, забезпечуючи актуальність інформації про стан периферійних пристроїв. Фрагмент програмного коду реалізації цього механізму наведено у лістингу 3.2.

Лістинг 3.2 – Функція збереження до бази даних

```
const saveSensorData = async (data) => { const record = new SensorData({ deviceId: data.deviceId, value: data.value, type: data.type, createdAt: Date.now() }); await record.save(); return record; };
```

Важливою складовою програмного комплексу є функція аналізу показників. Отримані значення порівнюються із встановленими пороговими параметрами, після чого система визначає нормальний або аварійний режим роботи. Якщо зафіксовано перевищення допустимих значень, автоматично формується повідомлення про подію та створюється відповідний запис у журналі системи.

Приклад реалізації алгоритму аналізу показників наведено у лістингу 3.3.

Лістинг 3.3 – Функція аналізу показників датчиків

```
const checkSensorValues = (value) => {
    const MAX_VALUE = 100;
    if (value > MAX_VALUE) {
        return {
            status: 'warning',
            message: 'Перевищено допустиме значення'
        };    }
    return {
        status: 'normal',
        message: 'Показники в нормі'
    };
};
```

Для відображення інформації у користувацькому інтерфейсі реалізовано функцію формування статистичних даних. Вона здійснює агрегацію інформації, сортування записів та підготовку масивів даних для побудови таблиць, графіків і діаграм. Завдяки цьому користувач отримує актуальну інформацію про стан міської інфраструктури у зручному для аналізу вигляді.Частина програмного коду відобраджена в додатку Б

Окремий модуль забезпечує роботу системи сповіщень. У разі виникнення критичної ситуації програмний комплекс автоматично формує повідомлення та

передає його адміністраторам або відповідальним операторам. Це дозволяє своєчасно реагувати на аварійні події та мінімізувати можливі наслідки.

Для підвищення швидкодії програмного комплексу реалізовано оптимізовані алгоритми виконання запитів до бази даних, використання асинхронної обробки інформації та кешування часто використовуваних даних. Застосування таких методів дозволяє суттєво зменшити час обробки інформації та забезпечити стабільну роботу системи при одночасному надходженні великої кількості повідомлень від периферійних пристроїв [22].

Під час реалізації функцій використовувалися сучасні засоби програмування та модульний принцип побудови програмного забезпечення, що спрощує супровід системи та дозволяє без значних змін інтегрувати нові функціональні можливості.

Таким чином, реалізовані функції для обробки даних забезпечують повний цикл роботи програмного комплексу — від приймання інформації до її аналізу, збереження та відображення користувачеві. Використані алгоритми та програмні рішення забезпечують високу продуктивність, надійність і масштабованість системи периферійних та хмарних обчислень у розумних містах.

3.3 Тестування розробленого програмного комплексу

Тестування програмного комплексу є важливим етапом розробки, який дозволяє перевірити правильність роботи всіх функціональних модулів системи, виявити можливі помилки та підтвердити відповідність програмного забезпечення поставленим вимогам. Проведення тестування забезпечує оцінку стабільності роботи системи та її готовності до практичного використання.

У процесі тестування було перевірено функціонування основних модулів програмного комплексу, серед яких модуль авторизації користувачів, система керування периферійними пристроями, модуль отримання та обробки даних, система моніторингу показників, аналітичний модуль та система формування повідомлень про критичні події.

Під час перевірки роботи модуля авторизації було підтверджено коректність реєстрації користувачів, перевірки облікових даних та розмежування прав доступу відповідно до ролей користувачів. Усі операції виконувалися без помилок, а доступ до функціональних можливостей системи надавався відповідно до встановлених правил.

Тестування модуля моніторингу показало правильність отримання інформації від периферійних пристроїв, її передачі до серверної частини та відображення у користувацькому інтерфейсі. Отримані показники своєчасно оновлювалися та відображалися у вигляді таблиць і графіків.

Окремо було перевірено роботу бази даних. Проведені випробування підтвердили коректність операцій додавання, редагування, пошуку та видалення інформації. Усі дані успішно зберігалися та були доступними для подальшої обробки.

Також виконано тестування системи формування повідомлень про аварійні ситуації. При моделюванні перевищення допустимих значень контрольованих параметрів програмний комплекс автоматично створював повідомлення та відображав його у журналі подій, що підтверджує правильність роботи алгоритмів аналізу інформації.

Для перевірки інтерфейсу користувача було проведено тестування вебзастосунку на різних пристроях та у різних веббраузерах. Результати показали коректне відображення всіх елементів інтерфейсу, швидке завантаження сторінок та стабільну роботу системи незалежно від програмного середовища.

Результати роботи програмного комплексу під час тестування наведено на рисунках 3.2–3.7**,** де представлено головну сторінку системи, сторінку моніторингу, модуль керування периферійними пристроями, аналітичний модуль, журнал подій та сторінку налаштувань. Наведені приклади підтверджують працездатність програмного комплексу та коректність реалізації його основних функцій.

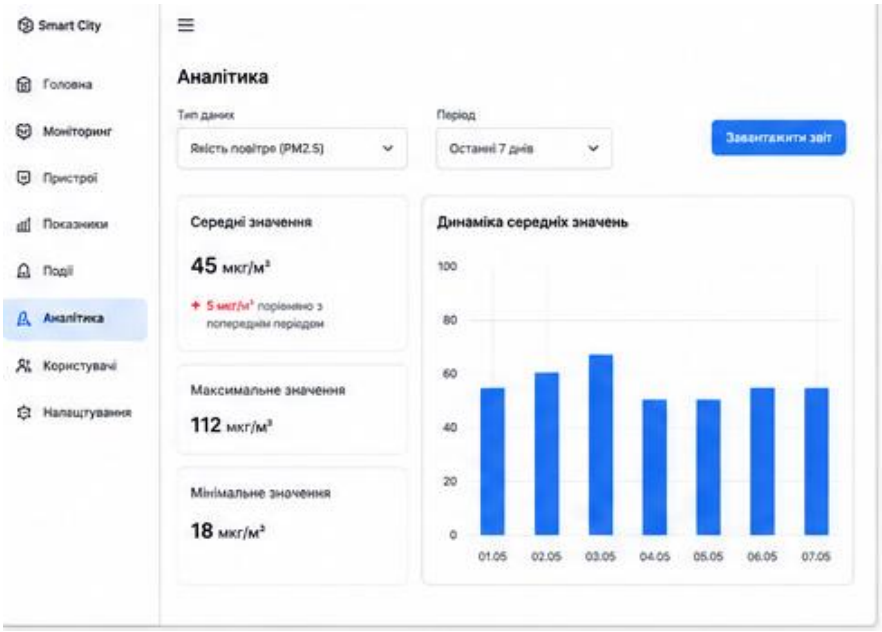


Рисунок 3.5 – Сторінка аналітики

На рисунку 3.5 представлено аналітичний модуль програмного комплексу, який забезпечує обробку статистичних даних та їх візуалізацію у вигляді графіків і діаграм. Це дозволяє аналізувати тенденції зміни показників та приймати обґрунтовані управлінські рішення.

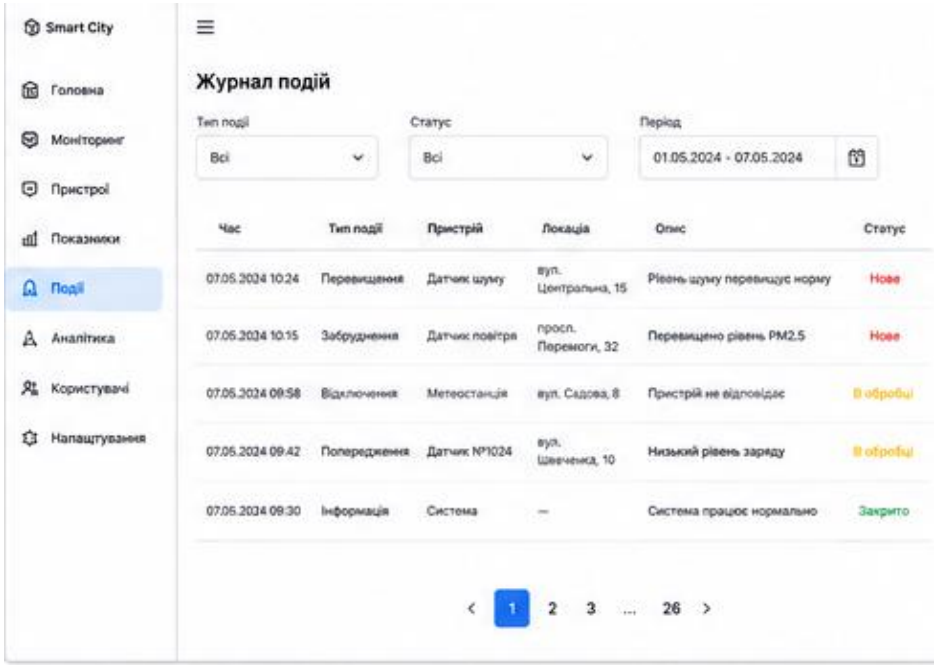


Рисунок 3.6 – Сторінка журналу подій

На рисунку 3.6 показано журнал подій, у якому відображаються всі зафіксовані системою повідомлення, аварійні ситуації та службові записи. Журнал забезпечує можливість контролю роботи програмного комплексу та аналізу історії подій.

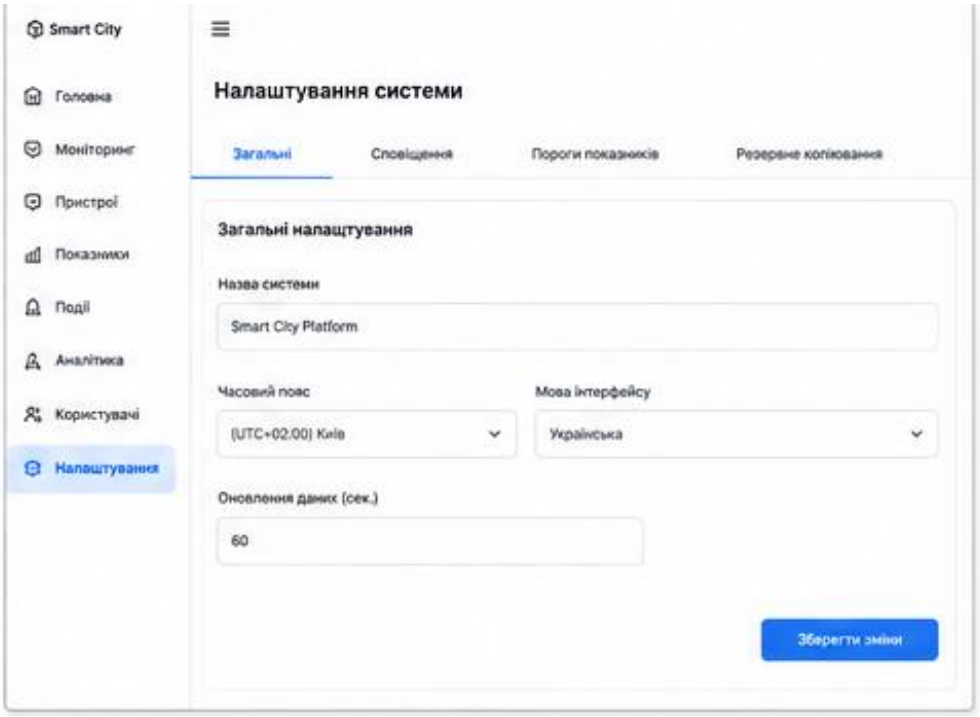


Рисунок 3.7 – Сторінка налаштувань системи

На рисунку 3.7 наведено сторінку налаштувань програмного комплексу, що призначена для конфігурації основних параметрів роботи системи та керування її функціональними можливостями. Використання цього модуля дозволяє адаптувати програмний комплекс до конкретних умов експлуатації

Отже, проведене тестування підтвердило правильність функціонування розробленого програмного комплексу, його відповідність поставленим вимогам, стабільність роботи та можливість використання для реалізації систем периферійних та хмарних обчислень у розумних містах.

3.4 Перспективи модернізації

Розроблений програмний комплекс для підтримки периферійних та хмарних обчислень у розумному місті виконує основні поставлені функції, однак у майбутньому його можна вдосконалити та розширити. Подальша модернізація системи дозволить підвищити її продуктивність, зручність використання, надійність та ефективність обробки даних [23].

Одним із перспективних напрямів розвитку є розширення переліку підтримуваних периферійних пристроїв. У майбутньому до системи можна інтегрувати додаткові датчики якості повітря, шуму, освітлення, руху транспорту, заповненості паркувальних місць та енергоспоживання. Це дозволить отримувати більш повну інформацію про стан міської інфраструктури.

Також доцільним є впровадження модуля прогнозування, який буде використовувати накопичені дані для аналізу тенденцій та передбачення можливих проблемних ситуацій. Наприклад, система зможе прогнозувати підвищення рівня забруднення повітря, перевантаження транспортних потоків або виникнення аварійних ситуацій на основі попередніх показників.

Перспективним напрямом модернізації є використання алгоритмів машинного навчання для автоматичного виявлення аномалій у даних. Це дозволить системі самостійно визначати нетипові зміни показників та швидше повідомляти операторів про можливі загрози [24], [25].

Ще одним напрямом удосконалення є розширення функціональності користувацького інтерфейсу. Зокрема, можна додати інтерактивну карту міста з відображенням усіх підключених пристроїв, розширену систему фільтрів, можливість формування звітів за вибраний період та експорт даних у різних форматах [26].

Для підвищення надійності роботи програмного комплексу доцільно реалізувати механізми резервного копіювання даних, автоматичного відновлення після збоїв та розподіленого зберігання інформації. Це забезпечить

стабільну роботу системи навіть у разі високого навантаження або технічних несправностей.

Також перспективним є впровадження мобільного застосунку для операторів та адміністраторів системи. Це дозволить здійснювати моніторинг стану міської інфраструктури, отримувати сповіщення та реагувати на критичні події незалежно від місця перебування користувача [27].

Отже, розроблений програмний комплекс має значний потенціал для подальшого розвитку. Його модернізація може бути спрямована на розширення кількості підтримуваних пристроїв, впровадження інтелектуальної аналітики, підвищення рівня безпеки, покращення інтерфейсу та інтеграцію з іншими міськими інформаційними системами.

3.5 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи було розглянуто практичні аспекти реалізації програмного комплексу для підтримки периферійних та хмарних обчислень у розумних містах. На основі попередньо виконаного проєктування здійснено розробку основних програмних модулів системи та визначено принципи їх взаємодії.

У ході роботи було спроєктовано інтерфейс користувача, який забезпечує зручний доступ до основних функцій програмного комплексу та ефективний моніторинг стану міської інфраструктури. Реалізовано сучасний вебінтерфейс, що дозволяє переглядати інформацію про периферійні пристрої, аналізувати отримані показники та здійснювати керування окремими елементами системи.

Також було розроблено основні функції для обробки даних, які забезпечують приймання інформації від IoT-пристроїв, її перевірку, аналіз, збереження у базі даних та формування повідомлень про критичні події. Використання модульного підходу дозволило забезпечити гнучкість програмного комплексу та можливість його подальшого розширення.

У процесі тестування підтверджено коректність роботи всіх основних функціональних модулів системи, правильність взаємодії між клієнтською та серверною частинами, а також стабільність роботи програмного комплексу при виконанні основних сценаріїв використання. Результати тестування засвідчили відповідність розробленої системи поставленим вимогам та готовність до практичного використання.

Крім того, було визначено перспективи подальшої модернізації програмного комплексу, які передбачають інтеграцію нових периферійних пристроїв, використання алгоритмів штучного інтелекту для аналізу даних, розширення аналітичних можливостей системи та впровадження мобільних сервісів для віддаленого керування.

Отже, виконані роботи з реалізації та тестування програмного комплексу підтвердили ефективність запропонованих методів і засобів периферійних та хмарних обчислень у системах розумного міста. Розроблений програмний комплекс є масштабованим, надійним і може слугувати основою для створення сучасних інформаційних систем моніторингу та керування міською інфраструктурою.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Менеджмент безпеки

Менеджмент безпеки є одним із ключових елементів сучасної системи управління підприємством, спрямованим на забезпечення безпечних умов праці, збереження життя і здоров'я працівників, а також запобігання виникненню небезпечних ситуацій під час виконання професійних обов'язків. Для підприємств, діяльність яких пов'язана з використанням інформаційних технологій та комп'ютерної техніки, питання організації безпечного робочого середовища є особливо актуальним, оскільки значна частина робочого часу проходить за персональними комп'ютерами.

Основною метою менеджменту безпеки є створення такої системи управління, яка дозволяє своєчасно виявляти потенційні небезпеки, оцінювати рівень професійних ризиків та впроваджувати комплекс організаційних, технічних і профілактичних заходів для їх мінімізації. При цьому увага приділяється не лише фізичній безпеці працівників, а й забезпеченню комфортних психофізіологічних умов праці, що безпосередньо впливають на продуктивність та якість виконуваних робіт. Основні складові системи менеджменту безпеки та їх взаємозв'язок наведено на рисунку 4.1.

Організація ефективного менеджменту безпеки передбачає розроблення внутрішніх нормативних документів, проведення інструктажів, навчання персоналу, контроль за дотриманням вимог охорони праці та регулярний аналіз можливих виробничих ризиків. Кожен працівник повинен бути ознайомлений із правилами безпечної експлуатації обладнання, порядком дій у разі виникнення аварійних ситуацій та основними вимогами пожежної безпеки.

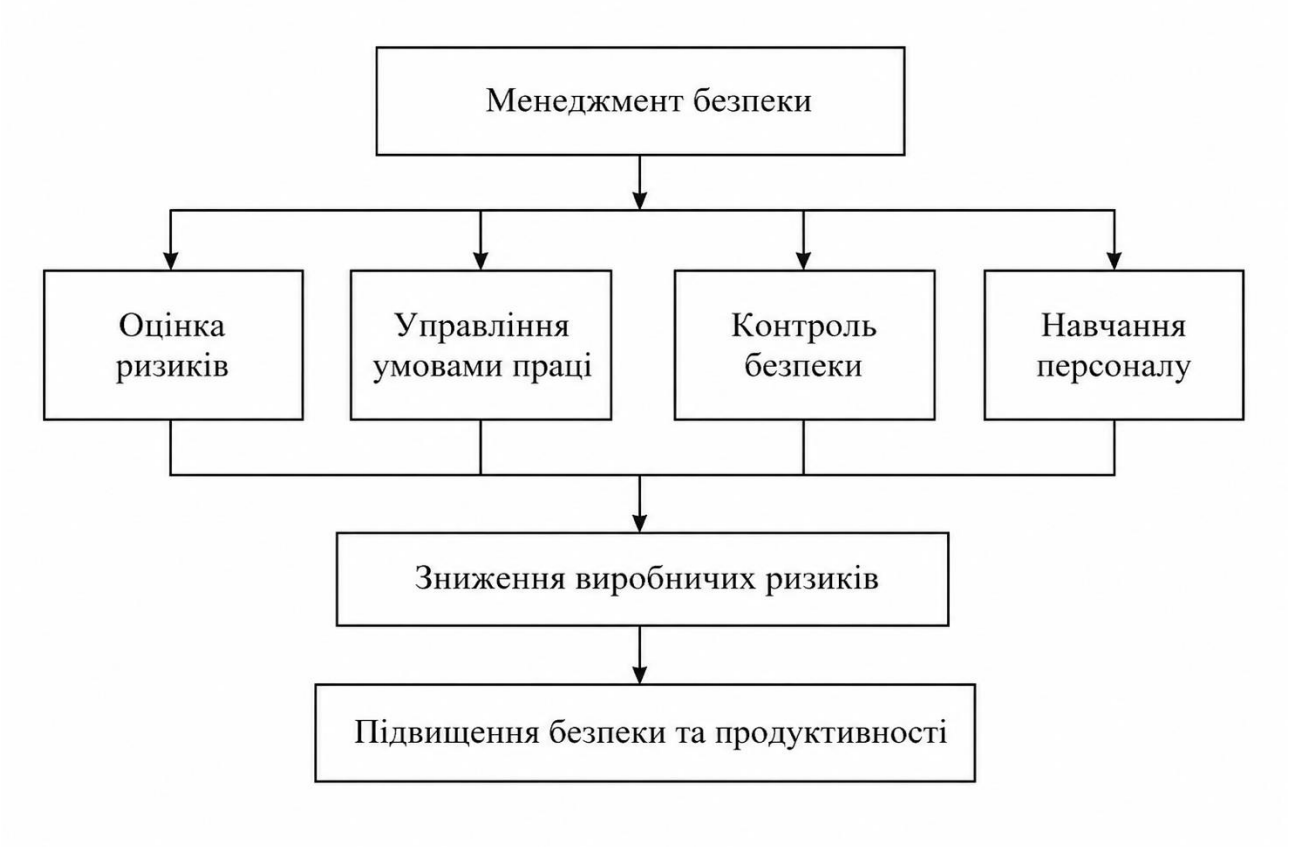


Рисунок 4.1 – Основні складові системи менеджменту безпеки на підприємстві

Під час роботи з комп'ютерною технікою основними небезпечними факторами є тривале статичне навантаження, перенапруження органів зору, недостатня рухова активність, психоемоційне навантаження та можливий вплив електричного струму у випадку несправності обладнання. Тривала робота без перерв може призводити до розвитку втоми, зниження концентрації уваги, погіршення самопочуття та збільшення кількості помилок під час виконання професійних завдань.

Важливою складовою менеджменту безпеки є правильна організація робочого місця відповідно до вимог ергономіки. Робочий стіл і крісло повинні забезпечувати правильне положення тіла працівника, монітор має бути розташований на оптимальній відстані від очей, а клавіатура та миша — у зручному положенні для рук. Особливу увагу необхідно приділяти освітленню робочої зони, яке повинно бути достатнім для комфортної роботи та не створювати відблисків на екрані монітора.

Не менш важливим фактором є підтримання оптимальних параметрів мікроклімату в приміщенні. Температура повітря, рівень вологості та швидкість руху повітря повинні відповідати санітарним нормам, що забезпечує комфортне перебування працівників протягом робочого дня. Регулярне провітрювання приміщень та використання систем кондиціонування позитивно впливають на працездатність персоналу та сприяють підтриманню належного рівня концентрації уваги.

Одним із головних напрямків менеджменту безпеки є управління професійними ризиками. Воно включає ідентифікацію небезпечних факторів, оцінку ймовірності їх виникнення, визначення можливих наслідків та розроблення заходів щодо їх усунення або зменшення впливу. Такий підхід дозволяє запобігати нещасним випадкам і створювати безпечне виробниче середовище.

Важливу роль у системі менеджменту безпеки відіграє навчання працівників. Регулярне проведення первинних, повторних та позапланових інструктажів дозволяє підтримувати необхідний рівень знань щодо правил охорони праці, пожежної безпеки та дій у надзвичайних ситуаціях. Працівники повинні знати порядок евакуації, місце розташування засобів пожежогасіння та правила користування ними [28], [29], [30].

Значний вплив на рівень безпеки має психологічний клімат у колективі. Сприятлива атмосфера, взаємоповага між працівниками та ефективна комунікація з керівництвом сприяють зниженню рівня стресу, покращують дисципліну праці та допомагають своєчасно виявляти потенційні небезпечні ситуації. Управління психоемоційними ризиками є невід'ємною складовою сучасного менеджменту безпеки.

Для підтримання високого рівня працездатності та профілактики професійної втоми доцільно організовувати короткі технологічні перерви кожні 1–2 години роботи за комп'ютером. Під час таких перерв рекомендується виконувати вправи для очей, змінювати положення тіла та здійснювати легку

фізичну розминку. Це сприяє покращенню кровообігу, зменшенню навантаження на хребет і м'язи та підтриманню високої концентрації уваги.

Ефективний менеджмент безпеки також передбачає постійний контроль технічного стану комп'ютерної техніки та електрообладнання. Регулярне технічне обслуговування, перевірка справності електромережі, використання джерел безперебійного живлення та захисних пристроїв дозволяють знизити ризик виникнення аварійних ситуацій та забезпечити безпечну експлуатацію обладнання [31].

З метою підвищення рівня безпеки на підприємстві рекомендується впроваджувати сучасні інформаційні системи моніторингу, автоматизовані засоби контролю виробничих процесів та електронні системи оповіщення про надзвичайні ситуації. Використання цифрових технологій дозволяє оперативно реагувати на потенційні загрози та підвищує ефективність управління безпекою [31].

Таким чином, система менеджменту безпеки є комплексом взаємопов'язаних організаційних, технічних та профілактичних заходів, спрямованих на створення безпечних умов праці та збереження здоров'я персоналу. Її ефективне функціонування забезпечує зниження виробничих ризиків, підвищення продуктивності праці, покращення якості виконання робіт та формування культури безпеки на підприємстві, що є важливою складовою успішної діяльності будь-якої сучасної організації [33].

4.2 Методи боротьби з монотонністю праці

Монотонність праці є одним із факторів, що негативно впливають на працездатність працівників, їхній психоемоційний стан та продуктивність виконання професійних обов’язків. Особливо це характерно для працівників, діяльність яких пов’язана з тривалою роботою за комп’ютером, виконанням однотипних операцій та постійною концентрацією уваги. Тривале виконання одноманітних дій призводить до швидкого розвитку втоми, зниження уважності,

погіршення швидкості прийняття рішень і збільшення ймовірності виникнення помилок.

Одним із найефективніших методів боротьби з монотонністю праці є раціональна організація режиму праці та відпочинку. Регламентовані короткочасні перерви дозволяють знизити фізичне та психічне навантаження, відновити працездатність і підтримувати високий рівень концентрації уваги протягом усього робочого дня. Під час перерв рекомендується змінювати вид діяльності, виконувати нескладні фізичні вправи або вправи для очей.

Важливу роль відіграє чергування різних видів робіт. Поєднання аналітичної діяльності, роботи з документацією, програмування, комунікації з колегами та інших завдань дозволяє уникнути тривалого виконання однотипних операцій і сприяє підтриманню інтересу до роботи. Такий підхід позитивно впливає на психологічний стан працівників і підвищує ефективність їхньої діяльності.

Для зменшення негативного впливу монотонності доцільно створювати комфортні умови праці. Ергономічне облаштування робочого місця, достатній рівень природного та штучного освітлення, оптимальні параметри температури й вологості повітря, а також мінімізація шумового навантаження сприяють збереженню працездатності та зменшенню відчуття втоми [33].

Позитивний вплив має також підтримка сприятливого психологічного клімату в колективі. Проведення спільних заходів, обмін досвідом між працівниками, можливість неформального спілкування та підтримка з боку керівництва сприяють емоційному розвантаженню і знижують рівень професійного стресу. Дружня атмосфера підвищує мотивацію до праці та формує позитивне ставлення до виконання службових обов'язків.

Одним із сучасних методів боротьби з монотонністю є використання цифрових інструментів планування робочого часу та систем управління завданнями. Вони допомагають рівномірно розподіляти навантаження, планувати перерви та контролювати виконання поставлених завдань, що сприяє підвищенню продуктивності праці та зменшенню перевтоми [34].

Не менш важливим є дотримання здорового способу життя. Повноцінний сон, збалансоване харчування, достатня фізична активність та регулярне перебування на свіжому повітрі позитивно впливають на функціональний стан організму та допомагають краще переносити інтелектуальні навантаження. Працівники, які дотримуються здорового способу життя, як правило, довше зберігають високу працездатність і менше схильні до професійного вигорання.

Отже, боротьба з монотонністю праці повинна здійснюватися комплексно та включати організаційні, ергономічні, психологічні й оздоровчі заходи [35]. Їх впровадження дозволяє підтримувати високий рівень працездатності працівників, зменшувати ризик виникнення професійної втоми та підвищувати ефективність роботи підприємства в цілому.

4.3 Висновок до четвертого розділу

У четвертому розділі було розглянуто основні аспекти охорони праці та менеджменту безпеки під час роботи з комп'ютерною технікою. Проаналізовано фактори, що впливають на безпеку та працездатність працівників, визначено значення правильної організації робочого місця, дотримання ергономічних вимог, забезпечення оптимального мікроклімату та виконання правил електро- і пожежної безпеки. Особливу увагу приділено системі менеджменту безпеки як комплексу організаційних і технічних заходів, спрямованих на зниження професійних ризиків та створення безпечних умов праці.

Також було досліджено методи боротьби з монотонністю праці, які передбачають раціональну організацію режиму праці та відпочинку, чергування різних видів діяльності, підтримання сприятливого психологічного клімату та використання сучасних інформаційних технологій для планування робочого процесу. Реалізація запропонованих заходів сприяє підвищенню продуктивності праці, зменшенню професійної втоми, покращенню самопочуття працівників і забезпечує ефективне та безпечне функціонування підприємства в умовах використання сучасних інформаційних технологій.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досліджено методи та засоби периферійних і хмарних обчислень у системах розумних міст та розроблено програмний комплекс, призначений для збору, обробки, аналізу й візуалізації інформації, що надходить від периферійних пристроїв міської інфраструктури. Запропоноване рішення забезпечує ефективний розподіл обчислювального навантаження між периферійними вузлами та хмарною платформою, що сприяє підвищенню швидкодії та надійності функціонування інформаційної системи.

У першому розділі кваліфікаційної роботи було проведено аналіз предметної області, розглянуто концепцію розумного міста та особливості використання технологій периферійних і хмарних обчислень. Визначено основні функціональні та нефункціональні вимоги до програмного комплексу, проаналізовано основних користувачів системи, обґрунтовано вибір програмних засобів і сучасних технологій, необхідних для реалізації поставлених завдань.

У другому розділі виконано проєктування програмного комплексу. Сформовано його структуру, розроблено архітектуру системи та модель взаємодії основних компонентів, спроєктовано структуру бази даних і алгоритм функціонування програмного забезпечення. Запропоновані проєктні рішення забезпечують ефективний обмін інформацією між периферійними пристроями, серверною частиною та хмарною інфраструктурою, а також створюють основу для масштабування системи.

У третьому розділі реалізовано програмний комплекс відповідно до розробленого проєкту. Створено користувацький інтерфейс, реалізовано функції приймання, обробки та аналізу інформації, а також механізми збереження даних і формування повідомлень про критичні події. Проведене тестування підтвердило коректність роботи всіх основних модулів системи, стабільність її функціонування та відповідність поставленим вимогам. Крім того, визначено перспективні напрями подальшої модернізації програмного комплексу.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці під час роботи з комп'ютерною технікою. Проаналізовано основні аспекти охорони праці та менеджменту безпеки, також було досліджено методи боротьби з монотонністю праці, наведено основні вимоги щодо організації безпечного робочого місця та рекомендації з дотримання норм охорони праці під час експлуатації інформаційних систем.

Отже, поставлена мета кваліфікаційної роботи була повністю досягнута, а всі визначені завдання успішно виконані. Розроблений програмний комплекс може бути використаний як основа для побудови сучасних інформаційних систем моніторингу та керування інфраструктурою розумних міст. Запропоновані методи та засоби периферійних і хмарних обчислень забезпечують ефективну обробку великих обсягів даних, підвищують оперативність прийняття управлінських рішень та створюють широкі можливості для подальшого розвитку концепції розумних міст.

ПЕРЕЛІК ДЖЕРЕЛ

1 Батура Т. В. Інтернет речей та концепція Smart City : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2022. 284 с.

2 Albino V., Berardi U., Dangelico R. M. Smart Cities: Definitions, Dimensions, Performance, and Initiatives. *Journal of Urban Technology*. 2015. Vol. 22(1). P. 3–21.

3 Smart Cities World. URL: https://www.smartcitiesworld.net/ (дата звернення: 10.02.2026).

4 Internet of Things (IoT). URL: https://www.ibm.com/topics/internet-of-things (дата звернення: 10.02.2026).

5 Microsoft Azure IoT. URL: https://azure.microsoft.com/services/iot/ (дата звернення: 10.02.2026).

6 AWS IoT Core. URL: https://aws.amazon.com/iot-core/ (дата звернення: 10.02.2026).

7 Google Cloud Smart Analytics. URL: https://cloud.google.com/ (дата звернення: 11.02.2026).

8 Cisco Smart Cities. URL: https://www.cisco.com/ (дата звернення: 11.02.2026).

9 OpenStreetMap Wiki URL: https://wiki.openstreetmap.org/ (дата звернення: 12.02.2026).

10 Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*. 2016.

11 Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*. 2016. Vol. 3(5). P. 637–646.

12 Satyanarayanan M. The Emergence of Edge Computing. *Computer*. 2017. Vol. 50(1). P. 30–39.

13 Mell P., Grance T. *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology, 2011.

14 Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 14.02.2026).

15 MongoDB Documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 14.02.2026).

16 Open Data Handbook. URL: https://opendatahandbook.org/ (дата звернення: 14.02.2026).

17 FIWARE. Open Source Platform for Smart Cities. URL: https://www.fiware.org/ (дата звернення: 14.02.2026).

18 OpenStack Documentation. URL: https://docs.openstack.org/ (дата звернення: 15.02.2026).

19 Kubernetes Documentation URL: https://kubernetes.io/docs/ (дата звернення: 15.02.2026).

20 NIST. Cybersecurity Framework. URL: https://www.nist.gov/cyberframework (дата звернення: 15.02.2026).

21 European Commission. Smart Cities Marketplace. URL: https://smart-cities-marketplace.ec.europa.eu/ (дата звернення: 10.02.2026).

22 Apache Kafka Documentation. URL: https://kafka.apache.org/documentation/ (дата звернення: 15.02.2026).

23 PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 15.02.2026).

24 Prometheus Documentation. URL: https://prometheus.io/docs/ (дата звернення: 16.02.2026).

25 OpenAPI Specification. URL: https://spec.openapis.org/oas/latest.html (дата звернення: 16.02.2026).

26 OpenCV Documentation. URL: https://docs.opencv.org/ (дата звернення: 16.02.2026).

27 European Telecommunications Standards Institute (ETSI). Multi-access Edge Computing (MEC). URL: https://www.etsi.org/technologies/multi-access-edge-computing (дата звернення: 16.02.2026).

28 Бедрій Я. І. Безпека життєдіяльності : навчальний посібник. Київ : Кондор, 2009. 286 с.

29 Працездатність людини-оператора. StudFiles. URL: https://studfile.net/preview/4474247/page:11/ (дата звернення: 05.06.2026).

30 Грибан В. Г., Негодченко О. В. *Охорона праці.* Київ : Центр учбової літератури, 2009. 209 с.

31 Геврик Є.О. Охорона праці. – К.: Ельга, Ніка-Центр, 2003, 280 с.

32 Пістун І.П. Безпека життєдіяльності. Навчальний посібник. – Суми; вид. «Університет кн.», 2000, 301 с.

33 ДСТУ 7234:2011. Дизайн і ергономіка. Обладнання виробниче. Загальні вимоги дизайну та ергономіки. Чинний від 01.08.2011. Вид. офіц. Український НДІ дизайну та ергономіки НАУ.

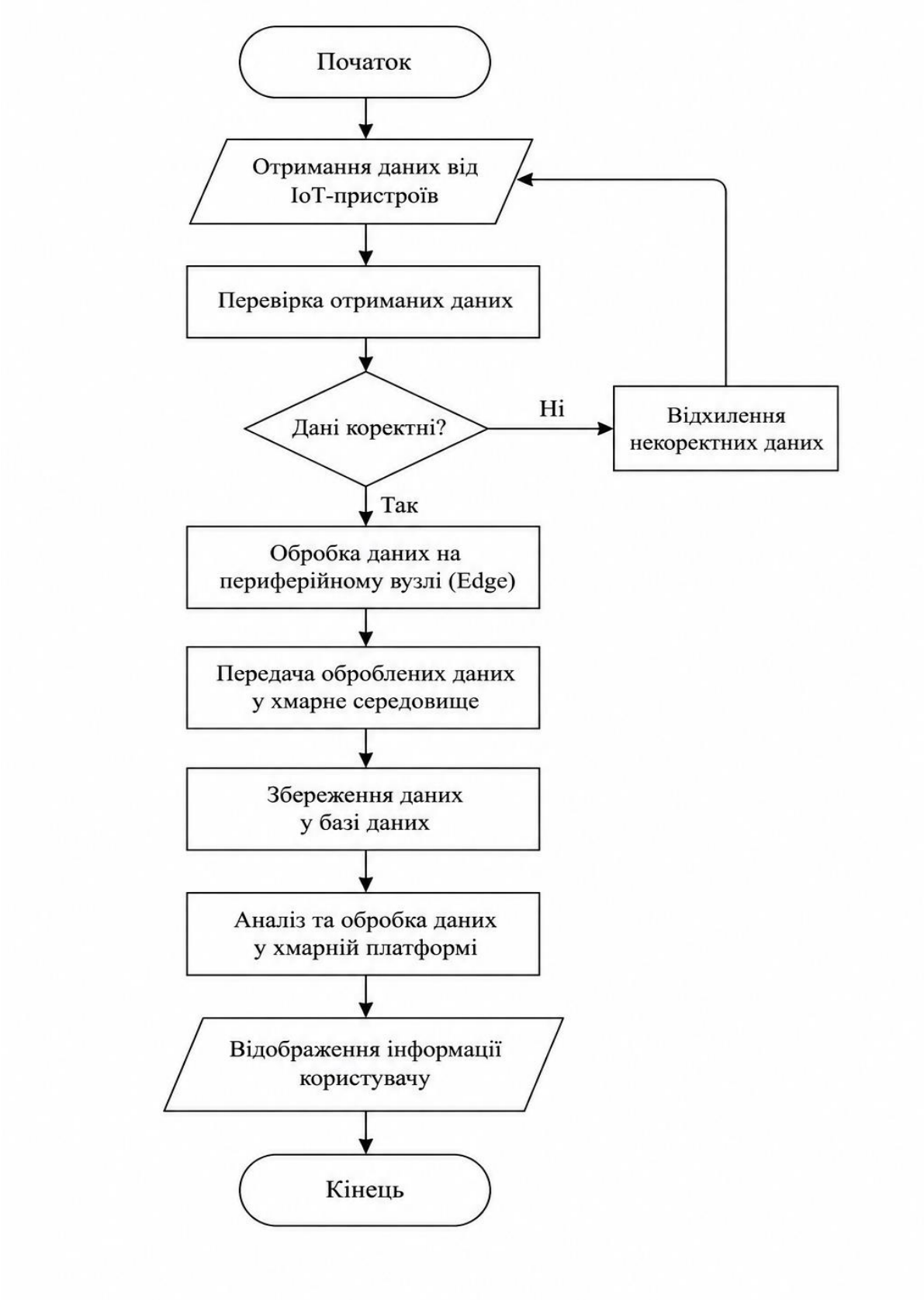
34 ДСТУ 7299:2013. Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки. Чинний від 01.01.2014. Вид. офіц.

35 Інструкція з охорони праці при роботі з комп'ютером, принтером, ксероксом та іншою оргтехнікою. Інструкції для навчальних закладів України. URL: https://osvita-docs.com/node/41 (дата звернення: 08.06.2026).

ДОДАТКИ

Додаток А

Блок-схема функціонування програмного комплексу



Додаток Б

Програмний код веб платформи

```
import os, textwrap, json, zipfile, pathlib, shutil

base = "/mnt/data/smart_city_platform"
if os.path.exists(base):
    shutil.rmtree(base)
os.makedirs(base)

files = {
"backend/package.json": """{
  "name": "smart-city-backend",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "nodemon server.js",
    "start": "node server.js",
    "seed": "node seed.js"
  },
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "mongoose": "^8.9.5"
  },
  "devDependencies": {
    "nodemon": "^3.1.9"
  }
}
""",
"backend/.env.example": """PORT=5000
MONGO_URL=mongodb://127.0.0.1:27017/smart_city_platform
""",
"backend/server.js": r"""
import express from "express";
import cors from "cors";
import mongoose from "mongoose";
import dotenv from "dotenv";

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

const PORT = process.env.PORT || 5000;
const MONGO_URL = process.env.MONGO_URL ||
"mongodb://127.0.0.1:27017/smart_city_platform";

mongoose.connect(MONGO_URL)
```

```
      .then(() => console.log("MongoDB connected"))
      .catch((err) => console.error("MongoDB error:",
err.message));

    const deviceSchema = new mongoose.Schema({
      name: String,
      type: String,
      location: String,
      status: { type: String, enum: ["online", "offline"],
default: "online" },
      lastUpdate: { type: Date, default: Date.now }
    });

    t", eventSchema);
    const Settings = mongoose.model("Settings", settingsSchema);

    app.get("/api/dashboard", async (req, res) => {
      const devices = await Device.find().sort({ lastUpdate: -1
});
      const activeDevices = devices.filter(d => d.status ===
"online").length;
      const readingsCount = await Reading.countDocuments();
      const criticalEvents = await Event.countDocuments({ status:
{ $in: ["Нове", "В обробці"] } });
      const latestEvents = await Event.find().sort({ time: -1
}).limit(5);

      res.json({
        activeDevices,
        readingsCount,
        criticalEvents,
        systemHealth: 98,
        latestEvents
      });
    });

    app.get("/api/devices", async (req, res) => {
      const devices = await Device.find().sort({ lastUpdate: -1
});
      res.json(devices);
    });

    app.post("/api/devices", async (req, res) => {
      const device = await Device.create(req.body);
      res.status(201).json(device);
    });

    app.get("/api/readings", async (req, res) => {
      const readings = await Reading.find().sort({ createdAt: 1
}).limit(50).populate("deviceId");
      res.json(readings);
    });
```

```
app.post("/api/readings", async (req, res) => {
  const data = req.body;
  const reading = await Reading.create(data);

  if (data.pm25 > 80 || data.pm10 > 90 || data.co2 > 800) {
    const device = await Device.findById(data.deviceId);
    await Event.create({
      type: "Перевищення",
      deviceName: device?.name || "Невідомий пристрій",
      location: device?.location || "-",
      description: "Показники перевищують допустиму норму",
      status: "Нове"
    });
  }

  res.status(201).json(reading);
});

app.get("/api/events", async (req, res) => {
  const events = await Event.find().sort({ time: -1 });
  res.json(events);
});

app.get("/api/analytics", async (req, res) => {
  const readings = await Reading.find().sort({ createdAt: -1
}).limit(30);
  const values = readings.map(r => r.pm25 || 0);
  const avg = values.length ? Math.round(values.reduce((a, b)
=> a + b, 0) / values.length) : 0;
  const max = values.length ? Math.max(...values) : 0;
  const min = values.length ? Math.min(...values) : 0;

  res.json({
    avg,
    max,
    min,
    chart: readings.reverse().map((r, index) => ({
      label: `${index + 1}`,
      value: r.pm25 || 0
    }))
  });
});

app.get("/api/settings", async (req, res) => {
  let settings = await Settings.findOne();
  if (!settings) settings = await Settings.create({});
  res.json(settings);
});

app.put("/api/settings", async (req, res) => {
  let settings = await Settings.findOne();
  if (!settings) settings = await Settings.create(req.body);
```

```
      else settings = await
Settings.findByIdAndUpdate(settings._id, req.body, { new: true });
      res.json(settings);
    });

    app.listen(PORT, () => console.log(`Server started on
http://localhost:${PORT}`));
    """,
    "backend/seed.js": r"""
    import mongoose from "mongoose";
    import dotenv from "dotenv";
    dotenv.config();

    const MONGO_URL = process.env.MONGO_URL ||
"mongodb://127.0.0.1:27017/smart_city_platform";

    const deviceSchema = new mongoose.Schema({
      name: String,
      type: String,
      location: String,
      status: String,
      lastUpdate: Date
    });

    const readingSchema = new mongoose.Schema({
      deviceId: { type: mongoose.Schema.Types.ObjectId, ref:
"Device" },
      pm25: Number,
      pm10: Number,
      co2: Number,
      temperature: Number,
      humidity: Number,
      createdAt: Date
    });

    const eventSchema = new mongoose.Schema({
      time: Date,
      type: String,
      deviceName: String,
      location: String,
      description: String,
      status: String
    });

    const settingsSchema = new mongoose.Schema({
      systemName: String,
      timezone: String,
      language: String,
      refreshSeconds: Number
    });

    const Device = mongoose.model("Device", deviceSchema);
    const Reading = mongoose.model("Reading", readingSchema);
```

```
    const Event = mongoose.model("Event", eventSchema);
    const Settings = mongoose.model("Settings", settingsSchema);

    await mongoose.connect(MONGO_URL);
    await Device.deleteMany();
    await Reading.deleteMany();
    await Event.deleteMany();
    await Settings.deleteMany();

    const devices = await Device.insertMany([
      { name: "Датчик якості повітря", type: "Сенсор", location:
"вул. Центральна, 15", status: "online", lastUpdate: new Date() },
      { name: "Датчик шуму", type: "Сенсор", location: "просп.
Перемоги, 32", status: "online", lastUpdate: new Date() },
      { name: "Метеостанція", type: "Сенсор", location: "вул.
Садова, 8", status: "online", lastUpdate: new Date() },
      { name: "Датчик паркування", type: "Сенсор", location:
"вул. Шевченка, 10", status: "offline", lastUpdate: new Date() },
      { name: "Датчик освітлення", type: "Сенсор", location:
"вул. Миру, 21", status: "online", lastUpdate: new Date() }
    ]);

    const readings = [];
    for (let i = 0; i < 30; i++) {
      readings.push({
        deviceId: devices[0]._id,
        pm25: Math.floor(30 + Math.random() * 75),
        pm10: Math.floor(35 + Math.random() * 55),
        co2: Math.floor(350 + Math.random() * 250),
        temperature: +(20 + Math.random() * 6).toFixed(1),
        humidity: Math.floor(40 + Math.random() * 20),
        createdAt: new Date(Date.now() - (30 - i) * 3600000)
      });
    }
    await Reading.insertMany(readings);

    await Event.insertMany([
      { time: new Date(), type: "Перевищення", deviceName:
"Датчик шуму", location: "вул. Центральна, 15", description:
"Рівень шуму перевищує норму", status: "Нове" },
      { time: new Date(), type: "Забруднення", deviceName:
"Датчик повітря", location: "просп. Перемоги, 32", description:
"Перевищено рівень PM2.5", status: "Нове" },
      { time: new Date(), type: "Відключення", deviceName:
"Метеостанція", location: "вул. Садова, 8", description: "Пристрій
не відповідає", status: "В обробці" },
      { time: new Date(), type: "Попередження", deviceName:
"Датчик №1024", location: "вул. Шевченка, 10", description:
"Низький рівень заряду", status: "В обробці" },
      { time: new Date(), type: "Інформація", deviceName:
"Система", location: "-", description: "Система працює нормально",
status: "Закрито" }
    ]);
```

```
  await Settings.create({
    systemName: "Smart City Platform",
    timezone: "(UTC+02:00) Київ",
    language: "Українська",
    refreshSeconds: 60
  });

  console.log("Seed completed");
  await mongoose.disconnect();
  """,
  "frontend/package.json": """{
    "name": "smart-city-frontend",
    "version": "1.0.0",
    "type": "module",
    "scripts": {
      "dev": "vite --host 0.0.0.0",
      "build": "vite build",
      "preview": "vite preview"
    },
    "dependencies": {
      "@vitejs/plugin-react": "^4.3.4",
      "axios": "^1.7.9",
      "vite": "^6.0.7",
      "react": "^18.3.1",
      "react-dom": "^18.3.1",
", icon: BarChart3 },
    { key: "events", label: "Події", icon: Bell },
    { key: "users", label: "Користувачі", icon: Users },
    { key: "settings", label: "Налаштування", icon: Settings }
  ];

  function Layout({ page, setPage, children }) {
    return (
      <div className="app">
        <aside className="sidebar">
          <div className="logo">🏙 Smart City</div>
          <nav>
            {nav.map(item => {
              const Icon = item.icon;
              return (
oї" value={data.activeDevices} note="↑ 12 за добу"/>
          <Card title="Поточні показники"
value={data.readingsCount} note="↑ 18 за добу"/>
          <Card title="Критичні події"
value={data.criticalEvents} note="↑ 2 за добу" danger/>
          <Card title="Працездатність системи"
value={`${data.systemHealth}%`} note="↑ 3%"/>
        </div>

        <div className="grid">
          <div className="panel">
            <h3>Карта міста</h3>
```

```
              <div className="map">
                <span className="pin green" style={{left:"20%",
top:"30%"}}></span>
                <span className="pin red" style={{left:"52%",
top:"45%"}}></span>
                <span className="pin yellow" style={{left:"70%",
top:"28%"}}></span>
                <span className="pin green" style={{left:"35%",
top:"68%"}}></span>
              </div>
            </div>

            <div className="panel">
              <h3>Останні події</h3>
              {data.latestEvents.map(e => (
                <div className="event" key={e._id}>
                  <span className={e.status === "Закрито" ? "dot
green" : e.status === "Нове" ? "dot red" : "dot yellow"}></span>
                  <div>
                    <b>{e.type}</b>
                    <p>{e.description}</p>
                  </div>
                </div>
              ))}
            </div>
          </div>
        </>
      );
    }

    function MonitoringPage() {
      const [readings, setReadings] = useState([]);

      useEffect(() => {
        axios.get(`${API}/readings`).then(res =>
setReadings(res.data));
      }, []);

      const latest = readings[readings.length - 1] || {};
      const max = Math.max(...readings.map(r => r.pm25 || 0), 1);

      return (
        <>
          <h1>Моніторинг показників</h1>
          <div className="filters">
            <select><option>Якість повітря
(PM2.5)</option></select>
            <select><option>Датчик №1003</option></select>
            <input type="date" defaultValue="2024-05-01"/>
            <button>Оновити</button>
          </div>

          <div className="panel">
```

```
            <h3>Графік зміни показників</h3>
            <div className="chart">
              {readings.map((r, i) => (
                <div
                  key={i}
                  className="bar"
                  style={{ height: `${((r.pm25 || 0) / max) *
220}px` }}
                  title={r.pm25}
                />
              ))}
            </div>
          </div>

          <div className="cards">
            <Card title="PM2.5" value={`${latest.pm25 || 0}
мкг/м³`} note="Норма"/>
            <Card title="PM10" value={`${latest.pm10 || 0}
мкг/м³`} note="Норма"/>
            <Card title="CO₂" value={`${latest.co2 || 0} ppm`}
note="Норма"/>
            <Card title="Температура"
value={`${latest.temperature || 0} °C`} note="Норма"/>
            <Card title="Вологість" value={`${latest.humidity ||
0}%`} note="Норма"/>
          </div>
        </>
      );
    }

    function DevicesPage() {
      const [devices, setDevices] = useState([]);

      useEffect(() => {
        axios.get(`${API}/devices`).then(res =>
setDevices(res.data));
      }, []);

      return (
        <>
          <h1>Керування пристроями</h1>
          <div className="filters">
            <input placeholder="Пошук пристрою..." />
            <select><option>Тип пристрою: всі</option></select>
            <select><option>Стан: всі</option></select>
            <button>Додати пристрій</button>
          </div>

          <div className="panel">
            <table>
              <thead>
                <tr>
            </table>
```

```
          </div>
        </>
      );
    }

    function AnalyticsPage() {
      const [analytics, setAnalytics] = useState(null);

      useEffect(() => {
        axios.get(`${API}/analytics`).then(res =>
setAnalytics(res.data));
      }, []);

      if (!analytics) return <p>Завантаження...</p>;

      return (
        <>
          <h1>Аналітика</h1>
          <div className="filters">
            <select><option>Якість повітря
(PM2.5)</option></select>
            <select><option>Останні 7 днів</option></select>
            <button>Завантажити звіт</button>
          </div>

          <div className="grid analytics">
            <div>
              <Card title="Середнє значення"
value={`${analytics.avg} мкг/м³`} note="порівняно з попереднім
періодом"/>
              <Card title="Максимальне значення"
value={`${analytics.max} мкг/м³`} note=""/>
              <Card title="Мінімальне значення"
value={`${analytics.min} мкг/м³`} note=""/>
            </div>
            <div className="panel">
              <h3>Динаміка середніх значень</h3>
              <div className="chart bars">
                {analytics.chart.map((item, i) => (
                  <div className="bar wide" key={i} style={{
height: `${item.value * 2}px` }}></div>
                ))}
              </div>
            </div>
          </div>
        </>
      );
    }
                <tr><th>Час</th><th>Тип
події</th><th>Пристрій</th><th>Локація</th><th>Опис</th><th>Статус
</th></tr>
              </thead>
              <tbody>
```

```
                {events.map(e => (
                  <tr key={e._id}>
                    <td>{new Date(e.time).toLocaleString("uk-
UA")}</td>
                    <td>{e.type}</td>
                    <td>{e.deviceName}</td>
                    <td>{e.location}</td>
                    <td>{e.description}</td>
                    <td>{e.status}</td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        </>
      );
    }

    function SettingsPage() {
      const [settings, setSettings] = useState(null);

      useEffect(() => {
        axios.get(`${API}/settings`).then(res =>
setSettings(res.data));
      }, []);
            <p>Модуль користувачів призначений для керування
операторами та адміністраторами системи.</p>
            <table>

<thead><tr><th>Ім’я</th><th>Роль</th><th>Статус</th></tr></thead>
              <tbody>

<tr><td>Оператор</td><td>operator</td><td>Активний</td></tr>

<tr><td>Адміністратор</td><td>admin</td><td>Активний</td></tr>
              </tbody>
            </table>
          </div>
        </>
      )
    }

    function App() {
      const [page, setPage] = useState("home");

      const pages = {
        home: <HomePage />,
        monitoring: <MonitoringPage />,
        devices: <DevicesPage />,
        analytics: <AnalyticsPage />,
        events: <EventsPage />,
        settings: <SettingsPage />,
        users: <UsersPage />
```

```
  };

  return (
    <Layout page={page} setPage={setPage}>
      {pages[page]}
    </Layout>
  );
}

createRoot(document.getElementById("root")).render(<App />);
""",
"frontend/src/style.css": r"""
* {
  box-sizing: border-box;
}

body {
  margin: 0;
  font-family: Arial, sans-serif;
  background: #f5f7fb;
  color: #111827;
}

.app {
  display: flex;
  min-height: 100vh;
}

.sidebar {
  width: 230px;
  background: #ffffff;
  border-right: 1px solid #e5e7eb;
  padding: 18px;
  display: flex;
  flex-direction: column;
  gap: 20px;
}

.logo {
  font-weight: 700;
  font-size: 18px;
  margin-bottom: 10px;
}

.sidebar nav {
  display: flex;
  flex-direction: column;
  gap: 8px;
}

.sidebar button {
  border: none;
  background: transparent;
```

```
  padding: 12px;
  border-radius: 10px;
  text-align: left;
  display: flex;
  gap: 10px;
  align-items: center;
  cursor: pointer;
  font-size: 15px;
}

.sidebar button.active,
.sidebar button:hover {
  background: #e8f1ff;
  color: #0d6efd;
}

.exit {
  margin-top: auto;
}

.content {
  flex: 1;
}

.topbar {
  height: 64px;
  background: #ffffff;
  border-bottom: 1px solid #e5e7eb;
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 16px 26px;
}

tart mongodb-community
```