Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи СУБД орієнтованої на IoT та Big Data”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Салтиков Г.В.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Коваленко А.С.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Салтикову Геннадію Вадимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи СУБД орієнтованої на IoT та Big Data*

2. Керівник роботи *Коваленко Анна Степанівна, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 47-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи СУБД орієнтованої на IoT та Big Data*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Коваленко А.С.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Салтиков Г.В.
(прізвище та ініціали)

АНОТАЦІЯ

Салтиков Г.В. Програмне забезпечення системи СУБД орієнтованої на IoT та Big Data. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи СУБД орієнтованої на IoT та Big Data.

Метою розробки є програмне забезпечення системи СУБД орієнтованої на IoT та Big Data.

Результат роботи – програмна реалізація системи СУБД орієнтованої на IoT та Big Data.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, СУБД, IoT, Big Data

ABSTRACT

Saltykov G.V. Software for a DBMS system focused on IoT and Big Data. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for a DBMS system focused on IoT and Big Data.

The purpose of the development is software for a DBMS system focused on IoT and Big Data.

The result of the work is a software implementation of a DBMS system focused on IoT and Big Data.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on a PC with OS Windows 10/11.

The program was developed in the Python environment.

Keywords: computer engineering, DBMS, IoT, Big Data

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АС	–	автоматизована система
АСДС	–	автоматизована система відповідних дослідницьких стендів
АСЕД	–	АС управління експериментальними дослідженнями
АСМ	–	автоматизована система моделювання гіпотетичних систем
АСНД	–	автоматизовані системи наукових досліджень
КМ	–	константи моделей
СВ	–	сигнали виміру
СМ	–	структури моделей
СУ	–	сигнали управління

ВСТУП

Актуальність теми. Унікальна по своїх можливостях СУБД орієнтована на IoT та Big Data легко справляється з обробкою даних не тільки бізнес-транзакцій, але також міжмашиної взаємодії й Інтернету речей у реальному часі.

Глобальна економіка входить в епоху Інтернету речей і масового міжмашиної взаємодії. Це значить, що вже незабаром, приблизно до 2027 року, по усьому світі прийдеться обробляти дані від 50 мільярдів смарт-пристроїв і одного трильйона додатків – усього близько 44 Збайт. Немає сумнівів, що колишні СУБД, орієнтовані на обробку транзакційних даних, що циркулюють у традиційних бізнес-додатках, не впораються з таким навантаженням. На зміну їм приходять СУБД нового покоління, споконвічно розраховані на роботу з великими обсягами й потоками даних.

Одна з них – СУБД орієнтована на IoT та Big Data, здатна аналізувати в реальному часі величезні обсяги інформації, одержуваної від усіляких «генераторів» даних – не тільки традиційних транзакційних систем, але також датчиків і пристроїв Інтернету речей, систем міжмашиної взаємодії, АСУТП, веб-сайтів і інших джерел.

По даним Gartner, у цей час СУБД орієнтована на IoT та Big Data лідирує серед аналітичних платформ по числу впроваджень із обсягом даних від сотень терабайт і більше.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи СУБД орієнтованої на IoT та Big Data.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем СУБД орієнтованої на IoT та Big Data.
– Дослідження системи СУБД орієнтованої на IoT та Big Data.
– Програмна реалізація системи СУБД орієнтованої на IoT та Big Data.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі СУБД орієнтованої на IoT та Big Data.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи СУБД орієнтованої на IoT та Big Data, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

В основу СУБД орієнтована на IoT та Big Data покладені наступні принципи.

– Виконання аналітичної (не транзакційної) обробки даних, причому й великі складні, і короткі аналітичні запити повинні оброблятися дуже швидко, у реальному часі.

– Архітектура платформи розрахована на масивно-паралельну обробку даних без використання поділюваних ресурсів (виключенням є мережа, що зв'язує елементи інфраструктури, на обчислювальних вузлах якої дані можуть оброблятися паралельно).

– Платформа лінійно масштабована, працююча на стандартному серверному встаткуванні архітектури x86.

– Забезпечується підтримка стандартної мови запитів SQL, застосовуваного в реляційних СУБД (тобто з погляду користувачів платформа виглядає як звичайна реляційна СУБД, але виконує аналітичні завдання).

– Платформа повинна підтримувати атомарність, цілісність, схоронність і ізольованість транзакцій – принцип ACID (Atomicity, Consistency, Isolation, Durability).

1.2 Область застосування

СУБД орієнтована на IoT та Big Data володіє чотирма унікальними властивостями:

– це по-справжньому колоночна СУБД (true column store);

– підтримуюча масивно-паралельну обробку даних (MPP);

– причому без використання поділюваних ресурсів (shared nothing);

– розширювана за рахунок підключення додаткових серійних серверів архітектури x86_64.

Зрозуміло, на ринку є й інші колоночні СУБД, що не використовують поділювані ресурси, а також СУБД, що підтримують MPP, але всіма чотирма перерахованими властивостями володіє тільки СУБД орієнтована на IoT та Big Data.

Перші три із чотирьох унікальних властивостей СУБД орієнтована на IoT та Big Data забезпечують її найвищу продуктивність, завдяки якій бізнес може одержувати самі актуальні дані й аналізувати інформацію в реальному часі. За допомогою MPP обробка розділяється між безліччю обчислювальних вузлів, кожний з яких виконує свою частину завдання. Відмова від застосування поділюваних ресурсів дозволяє уникнути вузьких місць в архітектурі, таких, наприклад, як очікування доступу до дискових систем. А завдяки колоночні архітектурі СУБД орієнтована на IoT та Big Data автоматично оптимізує фізичне зберігання даних, тобто фізичну модель даних, що дозволяє значно зменшити обсяги інформації, переданої в операціях читання з дисків (ці операції нерідко гальмують роботу СУБД), і домагатися високої продуктивності. Ще сильніше скоротити ці обсяги допомагає стиск даних.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи СУБД орієнтованої на IoT та Big Data, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Розмір глобального ринку великих даних

У 2024 році глобальний ринок великих даних становив 199,63 мільярда доларів США. Очікується, що розмір світового ринку зросте з 224,46 мільярда доларів США у 2025 році до 573,47 мільярда доларів США до 2033 року, зростаючи на середньорічному темпі зростання на 12,44% з 2025 по 2033 рік.

Великі дані містять великі обсяги даних, розмір яких (петабайти та ексабайти) перевищує можливості програмних засобів, які зазвичай використовуються для купівлі, керування та обробки даних з часом. Хоча великі дані не стосуються явних обсягів, тим не менш, цей термін широко використовується, коли мова йде про обробку великих обсягів даних. За останні роки екосистема великих даних та штучного інтелекту прискорилася з багатьма великими та малими компаніями. У результаті Індія стане одним з найбільших ринків великих даних і аналітики даних у всьому світі з покращеними сценаріями використання та значними можливостями для науковців з обробки даних у найближчі роки. Великі дані можуть дозволити підприємствам підвищити операційну ефективність і зменшити витрати. Багато компаній впроваджують рішення та послуги з використанням великих даних, щоб оцінити внутрішні процеси та покращити свою роботу. Впровадження Big Data допомагає компаніям знайти правильний баланс між операційними витратами, швидкістю, гнучкістю та якістю.

Глобальний ринок великих даних за останні роки значно зріс і, як очікується, буде зростати зі здоровим CAGR протягом прогнозованого періоду.

Багато організацій у різних галузях усвідомили потенціал аналізу даних і прийняли рішення для великих даних, щоб отримати перевагу даних. Збільшення попиту на рішення та послуги з великими даними останнім часом пов'язане насамперед із збільшенням використання рішень для великих даних для прийняття стратегічних рішень. Технологічні досягнення, такі як розповсюдження пристроїв Інтернету речей, удосконалення інфраструктури хмарних обчислень і алгоритмів машинного навчання, відіграли велику роль у прийнятті великих даних за останні кілька років. Використовуючи ці досягнення, кілька компаній збирали, зберігали та аналізували величезні обсяги даних і використовували отриману інформацію для розвитку своїх організацій.

Очікується, що протягом прогнозованого періоду глобальний ринок великих даних зазнає помітного зростання. Очікується, що швидке впровадження джерел даних, нових технологій, таких як штучний інтелект і периферійні обчислення, а також збільшення уваги до конфіденційності та безпеки даних підвищать попит на рішення та послуги для великих даних у всьому світі в найближчому майбутньому. Кілька компаній у всьому світі намагаються використовувати силу даних, щоб отримати конкурентну перевагу, що, як виявилося, сприяє зростанню глобального ринку великих даних.

Драйвери ринку

Значне збільшення споживчих і машинно-генерованих даних у всьому світі

Зростання глобального ринку великих даних в основному зумовлене значним збільшенням споживчих і машинно-генерованих даних у всьому світі. Щодня споживачі з усього світу генерують величезну кількість даних у формі публікацій у соціальних мережах, онлайн-транзакцій і даних датчиків із пристроїв Інтернету речей. З іншого боку, швидке впровадження підключених пристроїв і датчиків у різних галузях промисловості, починаючи від виробництва та охорони здоров'я до транспорту та роздрібної торгівлі, призводить до появи гігантського обсягу машинно-генерованих даних. Наприклад, за даними IDC, до 2025 року

глобальна сфера даних досягне 175 зетабайт. Згідно з джерелами, до 2027 року кількість підключених пристроїв у всьому світі перевищить 41 мільярд. Компанії в усьому світі знають потенціал даних, які генерують їхні споживачі та машини, і використовують їх для прийняття стратегічних рішень, застосовуючи рішення для великих даних.

Використання соціальних медіа та швидке впровадження IoT

Зростаюче використання соціальних медіа та швидке впровадження IoT у різних галузях промисловості по всьому світу, ймовірно, сприятиме зростанню глобального ринку великих даних. Кількість користувачів соціальних мереж у всьому світі стрімко зростає. Згідно з даними Statista, до 2025 року кількість користувачів соціальних мереж у всьому світі досягне 4,41 мільярда. Соціальні мережі щодня генерують величезні обсяги даних, які включають взаємодію користувачів, уподобання та настрої, а організації використовують дані, отримані своїми користувачами, щоб зрозуміти поведінку споживачів, тенденції та вподобання, що стимулює попит на послуги великих даних і сприяє зростанню світового ринку. Ринок IoT зростає значними темпами, що призвело до збільшення кількості інтелектуальних пристроїв, підключених через Інтернет, що, у свою чергу, сприяло вибуху даних. Відповідно до звіту Ericsson Mobility Report за 2018 рік, з 2018 по 2024 рік очікується, що загальне використання мобільних даних в Індії зросте на 26% у річному обчисленні.

Збільшення використання Інтернету

Зростання використання Інтернету та зростання обсягу даних, що генеруються в усьому світі, сприяють зростанню глобального ринку великих даних. Інтернет є основною причиною, чому ми переживаємо цей бум даних. Згідно зі звітом Cisco VNI, у 2022 році буде приблизно 4,8 мільярда користувачів Інтернету, або 60% населення світу. Зростання середньої швидкості Wi-Fi ще більше сприяє споживанню даних у всьому світі. Відповідно до прогнозу Cisco VNI Global IP Traffic Forecast, статистика споживання даних в Азіатсько-Тихоокеанському регіоні зросла на 100% у 2022 році порівняно з 2017 роком.

Потреба у великих даних розвинулася для керування системами керування реляційними базами даних, статистикою робочого столу та пакетами візуалізації для керування великими обсягами даних. Для великих організацій керування величезними обсягами даних вимагає ефективного рішення для керування даними.

Обмеження ринку

Конфіденційність

Занепокоєння щодо безпеки та конфіденційності даних, брак кваліфікованих спеціалістів у сфері великих даних, а також високі витрати на впровадження та обслуговування в першу чергу перешкоджають зростанню глобального ринку великих даних. Проблеми з інтеграцією різних джерел даних, обмежена обізнаність і розуміння великих даних, а також непостійна якість і цілісність даних гальмують темпи зростання світового ринку.

Ринкові можливості

Датчики та пристрої IoT

Очікується, що ринок великих даних досягне основних етапів у прогнозований період. Одними з ключових прихильників на цьому глобальному ринку є датчики та пристрої IoT. Згідно з дослідженнями, очікується, що світові дані досягнуть 175 зетабайт до 2025 року. Наприклад, стопка iPad щільністю 0,29 може збільшити відстань між Місяцем і Землею у 18 разів. Крім того, прогнозується, що в найближчі роки відбудеться величезне збільшення обсягу даних, тобто близько 60 відсотків великих даних для аналізу. Крім того, очікується, що на ринку 70 відсотків компаній перенесуть свій акцент на невеликі та широкі дані. Підприємства прагнуть бути менш залежними від штучного інтелекту для даних і надання фону для аналітики даних. Очікується, що компанії створюватимуть більш сильну аналітику даних, щоб гравці ринку були менш орієнтованими на інформацію та все ще надавали чудову інформацію з каналів неструктурованих даних.

Виклики ринку

Великі обсяги даних і зниження зручності використання даних

Постійно зростаючий обсяг і швидкість створюваних і збираних даних, обмежене різноманіття цієї інформації разом із попитом на більш точні та достовірні дані будуть ключовими проблемами для компаній, що працюють на цьому ринку. Проблеми включають великі обсяги даних, безпеку даних, зниження зручності використання даних і затримки, а також відсутність придатних і якісних даних перешкоджають розширенню ринку. Згідно з дослідженням 2023 року, великі дані більш схильні до кібератак або онлайн-атак, а середня світова вартість кожного витоку даних досягла 4,45 мільйона доларів США, що перевищує 4,35 мільйона доларів США у 2022 році. З появою великих даних відбувся еквівалентний сплеск хакерських здібностей або компетенції кібератак. Обхід звичайних кроків безпеки, таких як інструменти безпеки на основі підпису, більше не актуальний. Генерація підроблених даних є ще однією серйозною проблемою безпеки великих даних через їх застосування для маніпулювання та обману систем великих даних.

Сегментний аналіз

За типом Insights

За типом сегменту послуг належало 41,7% глобальної частки ринку великих даних у 2024 році, і очікується, що домінування продовжуватиметься протягом прогнозованого періоду. Зростання сегменту послуг пояснюється складністю проектів з великими даними, дефіцитом кваліфікованих фахівців і попитом на індивідуальні рішення, адаптовані до конкретних вимог бізнесу. Зростаюче впровадження хмарних аналітичних платформ і керованих сервісів для інтеграції даних і управління ще більше прискорює темпи зростання сегмента послуг на світовому ринку. Accenture, Deloitte та IBM є одними з головних гравців, які пропонують послуги великих даних організаціям у всьому світі.

Сегмент програмного забезпечення був другим за величиною сегментом і, за оцінками, зростатиме з багатообіцяючим CAGR протягом прогнозованого періоду. Потреба в прийнятті рішень на основі даних, прогнозній аналітиці та аналізі в реальному часі в різних галузях значно зростає в усьому світі та сприяє розширенню сегменту програмного забезпечення. Поява хмарних платформ великих даних та інтеграція ШІ та технологій машинного навчання ще більше сприяють зростанню сегменту програмного забезпечення. Такі компанії, як Microsoft, Oracle і SAP, пропонують програмне забезпечення для великих даних, і ці гравці разом займають 50% частки світового ринку програмного забезпечення для великих даних.

Апаратне забезпечення зайняло значну частку світового ринку в 2023 році і, за прогнозами, буде зростати зі здоровим CAGR протягом прогнозованого періоду. Зростання апаратного сегменту в основному зумовлене поширенням пристроїв Інтернету речей (IoT) і периферійних обчислювальних технологій, а також зростанням попиту на високопродуктивні обчислювальні системи, рішення для зберігання даних і мережеве обладнання. Такі компанії, як Dell Technologies, Hewlett Packard Enterprise (HPE) і IBM, відіграють провідну роль на світовому ринку обладнання для великих даних.

Від Deployment Mode Insights

Виходячи з режиму розгортання, у 2023 році локальний сегмент мав 59,8% частки світового ринку, і очікується, що він буде зростати зі здоровим CAGR протягом прогнозованого періоду. Положення про конфіденційність даних, такі як GDPR у Європі та HIPAA у Сполучених Штатах, сприяють прийняттю локальних рішень серед організацій, які обробляють конфіденційну інформацію, що є одним із ключових факторів, що сприяють зростанню локального сегменту. Зростаюче використання локальних рішень організаціями, які працюють у банківській сфері, охороні здоров'я та уряді, сприяє розширенню локального сегменту на світовому ринку.

За оцінками, хмарний сегмент демонструватиме значний CAGR протягом прогнозованого періоду. Масштабованість і гнучкість хмарних платформ для швидкого розгортання та масштабування рішень для великих даних є одним із ключових факторів зростання хмарного сегменту. Amazon Web Services (AWS), Microsoft Azure і Google Cloud Platform (GCP) відіграють провідну роль на ринку хмарних великих даних.

За розміром організації Insights

Виходячи з розміру організації, сегмент великих підприємств домінував на світовому ринку великих даних у 2023 році з 66,7% світової частки ринку та, за оцінками, зростатиме зі значним CAGR протягом прогнозованого періоду. Великі підприємства використовують великі дані для різних цілей, включаючи оптимізацію операцій, покращення взаємодії з клієнтами та отримання інформації для стратегічного планування. Нормативні вимоги, такі як GDPR і CCPA, ще більше спонукають великі підприємства інвестувати в надійні рішення для управління даними та безпеки, що є одним із ключових факторів, що сприяють розширенню сегмента великих підприємств на світовому ринку. Великі підприємства з таких галузей, як фінанси, охорона здоров'я та роздрібна торгівля, найбільше витрачають на впровадження рішень для великих даних. Наприклад, приблизно 80% компаній зі списку Fortune 500 користуються перевагами рішень для великих даних для стимулювання зростання бізнесу.

Очікується, що сегмент МСП зростатиме з найшвидшим CAGR у 13,22% протягом прогнозованого періоду. Малі та середні підприємства (МСП) все частіше використовують рішення для великих даних, щоб отримати інформацію та підвищити ефективність роботи. Можна помітити високе впровадження рішень для великих даних серед малих і середніх підприємств на ринках, що розвиваються, таких як Індія, Китай і Бразилія.

Глобальний аналіз ринку великих даних за бізнес-функцією

Виходячи з бізнес-функції, операційний сегмент лідирував на ринку в 2023 році, на нього припадало 60,8% світової частки ринку, і очікується, що він

зареєструє найшвидший CAGR протягом прогнозованого періоду. Пристрої Інтернету речей (IoT) швидко зростають у всьому світі, і ці пристрої генерують величезні обсяги даних, які можна аналізувати для покращення робочих процесів. Цей фактор значною мірою сприяє розширенню операційного сегменту на глобальному ринку великих даних. Збільшення кількості досягнень у технологіях аналітики даних, таких як машинне навчання та штучний інтелект, які допомагають підприємствам отримувати корисну інформацію з великих наборів даних, ще більше прискорює темпи зростання операційного сегменту на глобальному ринку. Виробництво, транспорт і охорона здоров'я є провідними виробниками рішень для великих даних для покращення робочих процесів.

Очікується, що фінансовий сегмент спостерігатиме значний CAGR протягом прогнозованого періоду завдяки зростаючому використанню рішень для великих даних фінансовими установами, такими як банки, страхові компанії та інвестиційні компанії, для управління ризиками, виявлення шахрайства та персоналізації послуг. Зростаюча потреба в аналітиці в реальному часі та штучному інтелекті у фінансах ще більше стимулює зростання фінансового сегменту. Фінансові установи використовують новітні технології для прийняття швидших і точніших рішень у таких сферах, як кредитний рейтинг, виявлення шахрайства та алгоритмічна торгівля.

За Industry Verticals Insights

Базуючись на галузевих вертикалях, сегмент BFSI відігравав домінуючу роль на світовому ринку великих даних і займав 21,6% світової частки ринку в 2023 році. Зростаюча потреба в аналізі величезних обсягів транзакційних даних у режимі реального часу для виявлення шахрайства, зниження ризиків і персоналізації послуг для клієнтів головним чином стимулює зростання сегмента BFSI на світовому ринку.

Регіональний аналіз

У 2023 році Північна Америка мала 41,2% світової частки ринку і стала найбільш домінуючим регіональним сегментом на світовому ринку. Домінування

Північної Америки, ймовірно, збережеться протягом прогнозованого періоду завдяки присутності технологічних гігантів, таких як Google, Amazon і Microsoft, які пропонують надійні хмарні рішення для великих даних. Наявність добре налагодженої IT-інфраструктури в поєднанні зі сприятливими урядовими ініціативами, які сприяють цифровій трансформації, ще більше сприяють зростанню північноамериканського ринку великих даних. США відіграють життєво важливу роль на ринку Північної Америки та мають великі центри для рішень для великих даних у Кремнієвій долині та Сіетлі. У Північній Америці такі галузі, як BFSI, охорона здоров'я та технологія, є основними прихильниками рішень для великих даних.

Європа є ще одним найефективнішим регіональним сегментом на світовому ринку. У 2023 році він захопив 25,8% світового ринку та, за прогнозами, буде зростати зі значним CAGR протягом прогнозованого періоду. У Європі діють суворі правила захисту даних, такі як GDPR, які зосереджуються на важливості конфіденційності та безпеки даних, що стимулює використання рішень для великих даних організаціями та сприяє зростанню європейського ринку великих даних. Зростаюче впровадження розширеної аналітики в різних галузях промисловості Європи ще більше сприяє зростанню європейського ринку. В Європі Сполучене Королівство, Німеччина та Франція є лідерами в розвитку інновацій у сфері великих даних, і на ці країни, ймовірно, припадатиме основна частка європейського ринку протягом прогнозованого періоду.

Азіатсько-Тихоокеанський регіон є найприбутковішим регіоном і, за прогнозами, зареєструє найшвидший CAGR протягом прогнозованого періоду. Присутність швидко зростаючих економік, таких як Китай, Індія та країни Південно-Східної Азії, а також зростаюча кількість ініціатив урядів країн Азіатсько-Тихоокеанського регіону, які сприяють розвитку розумних міст, прискорюють темпи зростання ринку Азіатсько-Тихоокеанського регіону. Прогнозується, що Китай та Індія відіграватимуть провідну роль на ринку великих даних Азійсько-Тихоокеанського регіону.

Ключові гравці ринку

Компанії, які відіграють провідну роль на світовому ринку великих даних, включають Microsoft (США), Teradata (США), IBM (США), Oracle (США), SAS Institute (США), Google (США), Adobe (США), Talend (США), Qlik (США), TIBCO Software (США), Alteryx (США), Sisense (США), Informatica (США), Cloudera (США), Splunk (США)), Palantir Technologies (США), 1010data (США), Hitachi Vantara (США), Fusionex (Малайзія), Information Builders (США), AWS (США), SAP (Німеччина), Salesforce (США) США), Micro Focus (Великобританія), HPE (Сполучені Штати), MicroStrategy (США) і ThoughtSpot (США).

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – це потужна мова програмування, яка проста у вивченні. Він має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис і динамічна типізація Python разом з його інтерпретованим характером роблять його ідеальною мовою для створення сценаріїв і швидкої розробки додатків у багатьох сферах на більшості платформ.

Інтерпретатор Python і обширна стандартна бібліотека доступні у вихідному або двійковому вигляді для всіх основних платформ на веб-сайті Python https://www.python.org/ і можуть вільно поширюватися. Цей же сайт також містить дистрибутиви та вказівники на багато безкоштовних сторонніх модулів Python, програм і інструментів, а також додаткову документацію.

Інтерпретатор Python легко розширюється за допомогою нових функцій і типів даних, реалізованих у C або C++ (або інших мовах, які можна викликати з C). Python також підходить як мова розширення для налаштовуваних програм.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи СУБД орієнтованої на IoT та Big Data.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Ключові висновки:

– Технології великих даних обробляють величезні набори даних, щоб отримати інформацію, інтегруючи ШІ, машинне навчання та Інтернет речей.

– До галузей, які використовують великі дані, належать охорона здоров’я, фінанси, роздрібна торгівля, логістика, виробництво, освіта, готельна справа, маркетинг, медіа та уряд.

– Серед прикладів використання великих даних – моніторинг у реальному часі, персоналізація клієнтів, операційна ефективність, управління ризиками та прогнозне обслуговування.

Дані – це нова нафта. Тим не менш, компанії в усіх галузях все ще повинні перетворити його на корисні продукти, як нафта перетворюється на газ або пластик. Тобто підприємства стикаються з проблемами управління величезними обсягами неструктурованих і структурованих даних. У той час як традиційним системам важко обробляти складні набори даних, переваги великих даних у бізнесі дозволяють організаціям отримувати корисну інформацію, покращувати взаємодію з клієнтами та приймати обґрунтовані рішення.

Такі технології, як ШІ, машинне навчання, і IoT прискорюють цю трансформацію, пропонуючи аналіз даних у реальному часі та можливості прогнозування. У цій роботі ми поговоримо про випадки використання великих даних для різних галузей і успішні приклади компаній, які використовують великі дані.

Технології великих даних – це спеціалізовані інструменти, платформи та методи, які обробляють, зберігають та аналізують масивні та складні набори даних. Це дозволяє організаціям отримати професійний штучний інтелект

консультації, приймати зважені рішення та стимулювати інновації. Великі дані часто працюють із штучним інтелектом, машинне навчання та Інтернет речей.

Глобальний ринок аналітики великих даних проектується зросте з $199,63 млрд у 2024 році до $510,03 млрд до 2032 року, зростаючи на 12,44% CAGR. Популярні приклади технологій великих даних включають бази даних Hadoop, Apache Spark і NoSQL.

Тепер давайте дізнаємося, які галузі використовують великі дані.

Випадки використання великих даних за галузями

Прикладами великих даних у бізнесі є охорона здоров'я, банківська справа, роздрібна торгівля, транспорт і логістика, уряд, виробництво, освіта, маркетинг, медіа та розваги.

Звичайно, ці області зазнають різного ступеня технологічного впливу. Зокрема, коли йдеться про витрати, 18% технологічних компаній інвестувати більше 20% своїх бюджетів витрачають на інструменти аналітики. Серед інших галузей із значними інвестиціями – роздрібна торгівля, охорона здоров'я, фінансові послуги, засоби масової інформації, бізнес-послуги та юридичні послуги.

Але для чого використовується аналітика великих даних? Давайте розглянемо сфери, які отримують найбільшу користь від технологій, обговорюючи популярні варіанти використання та приклади великих даних у бізнесі.

Великі дані в охороні здоров'я

Технологія покращує якість медичної допомоги, знижує витрати та підтримує інновації за допомогою таких випадків використання великих даних:

– Діагностика. ШІ та великі дані підвищують точність діагностики (наприклад, у радіології або телемедицина) шляхом аналізу великих обсягів даних.

– Персоналізована медицина. Постачальники медичних послуг створюють індивідуальні плани лікування на основі індивідуальних потреб і умов

шляхом аналізу даних пацієнтів.

– Розробка ліків. Аналіз великого набору даних прискорює відкриття нових ліків і методів лікування шляхом виявлення перспективних сполук.

– Прогностична аналітика. Використання великих даних у бізнесі включає прогнозування тенденцій здоров’я пацієнтів, можливість раннього втручання та ефективний розподіл ресурсів.

Приклади великих даних в охороні здоров'я:

– Централізуючи дані та розвиваючи досвід ШІ, Pfizer зберігає мільйонів і прискорює інновації. Хмара наукових даних спрощує доступ до даних для вчених, а VOX на основі штучного інтелекту підвищує ефективність досліджень і постачання продукції. Крім того, великі дані та ШІ підтримали мету компанії випустити 19 ліків і вакцин за 18 місяців.

Великі дані в банківській справі

Застосування великих даних у фінансах допомагає установам оптимізувати процеси та підвищити ефективність.

– Виявлення шахрайства. Завдяки великим наборам даних фінансові служби виявляють підозрілі моделі та передбачають потенційне шахрайство.

– Кредитний скоринг. Компанії, які використовують великі дані, оцінюють кредитоспроможність, аналізуючи історію транзакцій та інші джерела даних (цифровий слід, платіжну поведінку, використання мобільного зв’язку тощо).

– Відповідність нормативним вимогам. Увімкніть відстеження транзакцій, ведення записів і моніторинг підозрілої активності для забезпечення відповідності.

– Алгоритмічний трейдинг. Банки та інвестиційні компанії можуть визначати ринкові тенденції та приймати торгові рішення в реальному часі на основі фінансових даних.

Приклади аналітики великих даних:

– JP Morgan Chase використовує великі дані, щоб виявити шахрайство

шляхом аналізу звичайних і шахрайських транзакцій. Симулятори AI генерують синтетичні дані з попередньо визначеними ймовірностями для виявлення аномалій.

Великі дані в роздрібній торгівлі

Давайте подивимося, як випадки використання роздрібної торгівлі великими даними сприяють більш ефективним, персоналізованим і орієнтованим на клієнта роздрібним операціям:

– Персоналізація. Роздрібні торговці використовують великі дані в промисловості для аналізу поведінки та вподобань клієнтів, що дозволяє персоналізувати рекомендації щодо продуктів.

– Динамічне ціноутворення. Здійснюйте коригування цін у режимі реального часу на основі попиту, цін конкурентів і ринкових тенденцій, щоб залишатися конкурентоспроможними, максимізуючи прибуток.

– Оптимізація запасів. Аналізуючи минулі продажі, сезонні тенденції та інші змінні, ви зможете передбачити моделі попиту та оптимізувати рівень запасів.

– Ефективність ланцюга поставок. Підприємства електронної комерції оцінюють дані в реальному часі від постачальників, транспортування та прогнозують попит, щоб зменшити витрати та скоротити час доставки.

Приклади великих даних у роздрібній торгівлі:

– Walmart використовує передові технології для створення персоналізованого досвіду покупок у магазинах, клубах Сема, веб-програмиі віртуальні середовища. Наприклад, платформа Content Decision Platform дозволяє створювати унікальні домашні сторінки та персоналізований контент.

Великі дані в транспорті та логістиці

Переваги великих даних у бізнесі включають підвищення ефективності, безпеки та досвіду клієнтів. Розглянемо ці важливі приклади використання аналітики великих даних:

– Оптимізація маршруту. Збирайте та використовуйте дані в режимі

реального часу з датчиків дорожнього руху, GPS і історичних тенденцій, щоб оптимізувати маршрути доставки та скоротити час у дорозі.

– Управління автопарком. Логістичні компанії оптимізують продуктивність автопарку, зменшують витрати та підвищують безпеку, відстежуючи показники автомобіля та водія.

– Оптимізація навантаження. Компанії, які використовують аналітику великих даних, оцінюють конфігурацію завантаження, щоб максимально використовувати простір у вантажівках і контейнерах.

– Логістична видимість. Відстеження продуктів у реальному часі покращує прозорість, зменшує затримки та покращує управління запасами.

Приклади з реального життя великі дані в логістиці:

– Maersk використовує великі дані для зменшення відходів, доступу до нових ринків і керування збоями. Його 700 суден генерують приблизно 5000 тегів даних кожне, а понад 70 портових споруд оснащено 750 000 пристроїв IoT.

Великі дані у виробництві

Галузі великих даних, такі як виробництво, мають революціонізувати виробничі процеси, підвищивши ефективність і якість продукції.

– Прогнозне обслуговування. Програми для великих даних допомагають скоротити час простою та витрати на технічне обслуговування, аналізуючи дані з датчиків обладнання.

– Прогнозування попиту. Оцінюючи клієнтські та ринкові дані, компанії прогнозують коливання попиту, щоб скорегувати графіки виробництва та зменшити перевиробництво або дефіцит.

– Контроль якості. Виявляйте дефекти на ранній стадії виробничого процесу, щоб швидко виправити проблеми та зменшити кількість браку.

– Оптимізація ланцюга поставок. Відстежуйте матеріали та продукти в режимі реального часу, забезпечуючи оптимальний рівень запасів, зменшуючи відходи та покращуючи графіки доставки.

Приклади аналітики великих даних у виробництві:

– Тесла перевизначає виробництво автомобілів, збираючи дані датчиків за мільйони миль для підвищення продуктивності, виявлення несправностей і вдосконалення функцій автономного водіння. Аналіз у режимі реального часу дозволяє оновлення по повітрю, покращуючи транспортні засоби та пропонуючи такі послуги, як дистанційна діагностика.

Великі дані в освіті

Освіта має бути більш ефективною та персоналізованою, і великі дані для промисловості можуть значною мірою сприяти цьому через:

– Персоналізоване навчання. Аналізуючи дані студентів, викладачі адаптують навчальний досвід відповідно до індивідуальних потреб і покращують залучення.

– Прогностична аналітика. Перехід до сучасної архітектури а також запровадити великі дані, щоб уможливити раннє втручання для студентів із групи ризику та підтримувати прийняття рішень на основі даних для адміністраторів.

– Утримання студентів. Освітні компанії, які використовують великі дані, виявляють студентів, які ризикують кинути навчання, щоб запровадити заходи.

– Розробка навчального плану. Вивчення тенденцій успішності студентів дозволяє навчальним закладам удосконалювати курси, усувати прогалини у змісті та створювати спеціальні навчальні матеріали.

Приклади застосування великих даних в освіті:

– Експертні оцінки Coursera впорядкували зворотній зв’язок із швидшим (1 хвилина проти 15 годин), детальнішим (у 45 разів більше відгуків) і масштабованим оцінюванням. Ранні тести показують збільшення кількості завершених курсів на 16,7%, тоді як оцінювання ШІ підтримує глибше навчання та утримання.

Аналітика великих даних у маркетингу

Чому аналітика великих даних важлива для маркетингу? Передові технології мають вирішальне значення для підвищення залученості клієнтів і

стимулювання зростання бізнесу. Розглянемо наступні випадки використання великих даних для бізнесу:

– Сегментація клієнтів. Створюйте цільові маркетингові стратегії та персоналізовані кампанії, точніше сегментуючи аудиторії.

– Оптимізація кампанії. Використовуючи вплив великих даних на бізнес, ви вдосконалите його стратегії в режимі реального часу, оптимізуючи витрати та максимізуючи віддачу від інвестицій.

– Прогностична аналітика. Партнер софшорні девелоперські компаніїщоб ефективно передбачати тенденції, оптимізувати запаси та підвищувати задоволеність клієнтів, пропонуючи відповідні продукти та послуги.

– Аналіз настроїв клієнтів. Вимірюйте настрої споживачів і коригуйте стратегії, щоб покращити сприйняття бренду.

Приклади маркетингових програм аналітики великих даних:

– Кока-Кола інтегрує дані про регіональних клієнтів в уніфіковану платформу, що дозволяє отримувати глобальну інформацію в реальному часі. Це дозволяє компанії розуміти глобальні тенденції та розробляти персоналізовані маркетингові стратегії на основі індивідуальних уподобань споживачів.

Великі дані в ЗМІ та розвагах

Думаєте про те, як покращити доставку вмісту та персоналізувати досвід? Випадки використання великих даних у ЗМІ та індустрії розваг оптимізують операційні процеси:

– Створення контенту. Індустрія аналітики великих даних розуміє тенденції та вподобання в реальному часі, дозволяючи творцям створювати контент, який резонує з аудиторією.

– Статистика аудиторії. Збираючи дані від глядачів на різних платформах, компанії отримують цінну інформацію про демографічні показники та поведінку аудиторії.

– Спеціальний контент. Проаналізуйте звички перегляду й уподобання, щоб створювати персоналізований контент, підвищуючи залученість і утримання.

– Аналітика живих подій.Інтегруйте рішення для великих данихдля оцінки даних у режимі реального часу з подій у прямому ефірі, включаючи настрої аудиторії, взаємодію в соціальних мережах і статистику глядачів.

Приклади великих даних в індустрії розваг:

– Netflix персоналізує рекомендації вмісту та мініатюри на основі звичок перегляду. Подібним чином Spotify використовує дані, щоб адаптувати музичні пропозиції для своїх246 мільйонів преміум-підписників, покращуючи взаємодію та задоволеність на різних платформах.

Застосування великих даних в державному та державному секторах

Більш чуйні послуги для громадян реальні. Все, що вам потрібно, це використовувати великі дані в різних галузях, як-от уряд і державний сектор.

– Розумні міста. Будувати SaaS MVP аналізувати дані датчиків і пристроїв Інтернету речей для оптимізації управління трафіком, енергоспоживанням і спеціальними системами громадського транспорту.

– Профілактика злочинності. Моделі злочинності, прогнозування злочинної діяльності та розподіл ресурсів є прикладами того, як великі дані допомагають знизити рівень злочинності та підвищити громадську безпеку.

– Охорона навколишнього середовища. Уряди співпрацюють з надійними постачальники великих даних для моніторингу змін навколишнього середовища, відстеження забруднення, прогнозування ризиків і забезпечення дотримання правил.

– Розподіл ресурсів і складання бюджету. Великі дані дозволяють урядам відстежувати витрати, оцінювати ефективність державних програм і виявляти області неефективності.

Приклади урядового застосування великих даних:

– Сінгапур джгути IoT для інновацій у сфері мобільності, житла, охорони здоров’я та управління, плавно інтегруючи цифрові та фізичні простори через 5G. У 2022 році його ринок Інтернету речей заробив 5,47 мільярда доларів США, причому 28% мешканців використовували пристрої Інтернету речей, що

закріпило його світове лідерство в цифрових інноваціях.

Отже, для чого можна використовувати великі дані? Давайте підсумуємо застосування великих даних у бізнесі в різних галузях:

– Моніторинг в реальному часі. Відстежуйте продуктивність, якість або надання послуг для миттєвого аналізу та коригування.

– Прогностична аналітика. Прогнозуйте тенденції, поведінку споживачів і ринкову динаміку для активного прийняття рішень.

– Персоналізація клієнта. Налаштуйте продукти, послуги та маркетингові стратегії на основі індивідуальних уподобань і поведінки.

– Оперативна ефективність. Оптимізуйте ланцюжки постачання, зменшіть витрати та покращте розподіл ресурсів.

– Управління ризиками. Визначайте та зменшуйте ризики, такі як шахрайство, загрози кібербезпеці та збої в роботі.

– Прогнозне обслуговування. Відстежуйте життєвий цикл і продуктивність фізичних активів, щоб зменшити час простою та витрати на обслуговування.

3.2 Розробка структурної схеми

Важлива перевага СУБД орієнтованої на IoT та Big Data – зручність розгортання: для роботи цієї СУБД не потрібний спеціалізований програмно-апаратний комплекс (appliance), вона прекрасно себе «почуває» на серійних 64-розрядних X 86-серверах на платформі Linux з локальними жорсткими дисками. Нагадаємо, СУБД орієнтована на IoT та Big Data ліцензується тільки як програмний продукт і може бути розгорнута на встаткуванні будь-якого вендора. Важливо відзначити, що витрати часу на адміністрування мінімальні.

Ще одна ключова перевага цієї СУБД – можливість зберегти з її допомогою як раніше зроблені, так і майбутні інвестиції. Оскільки СУБД орієнтована на IoT та Big Data підтримує стандартний ANSI SQL 99 і принцип

ACID, переучувати персонал не прийде, а крім того, удасться уникнути витрат на модифікацію інфраструктури додатків, що працюють із СУБД за допомогою SQL. За допомогою відкритих інтерфейсів СУБД орієнтована на IoT та Big Data легко вбудовується в навний аналітичний ландшафт при установці замість колишньої СУБД, що застосовувалася для аналітичних завдань. Що цікаво, кластер СУБД орієнтованої на IoT та Big Data можна плавно розширювати від терабайт до петабайт «сирих», тобто ліцензуємих, даних без принципової зміни його інфраструктури.

Схоронності інвестицій сприяють і правила ліцензування: один крок становить 1 Тбайт «сирих» даних, що дає можливість розширювати використання СУБД орієнтована на IoT та Big Data у міру нарощування обсягів інформації (до речі, цієї СУБД можна користуватися безкоштовно, якщо обсяг оброблюваних даних не перевищує 1 Тбайт), причому враховується загальний обсяг аналізованих даних. Якщо, приміром, крім продуктивного кластера СУБД орієнтованої на IoT та Big Data створюється тестові або резервний, додаткові ліцензії не будуть потрібні (при цьому, звичайно, загальний обсяг даних не повинен перевищувати показники, передбачені діючою ліцензією).

Нарешті, важливий той факт, що архітектура СУБД орієнтованої на IoT та Big Data забезпечує дуже гарний захист від різного роду збоїв. Функції реплікації, резервного копіювання й відновлення даних дозволяють створювати катастрофостійкі конфігурації з розміщенням «гарячих» кластерів у декількох вилучених один від іншого дата-центрах. Кластеризація забезпечує й захист від збоїв у ході лінійного масштабування цієї системи.

СУБД прекрасно справляється не тільки з однорідної, але й зі змішаним аналітичним навантаженням, коли одночасно можуть працювати додатка, орієнтовані на різні види аналітики – від нескладних звітів до глибокого дослідження даних і пошуку закономірностей у них. А завантаження даних у СУБД орієнтованої на IoT та Big Data можна здійснювати паралельно з їхнім аналізом.

СУБД орієнтовану на IoT та Big Data тісно інтегрована з Hadoop і легко справляється як зі структурованими, так і з напівструктурованими даними, такими, наприклад, як усілякі системні журнали (логи). Для аналізу мультимедійних даних HPE рекомендує застосовувати платформу IDOL, інтеграція з якої теж підтримується.

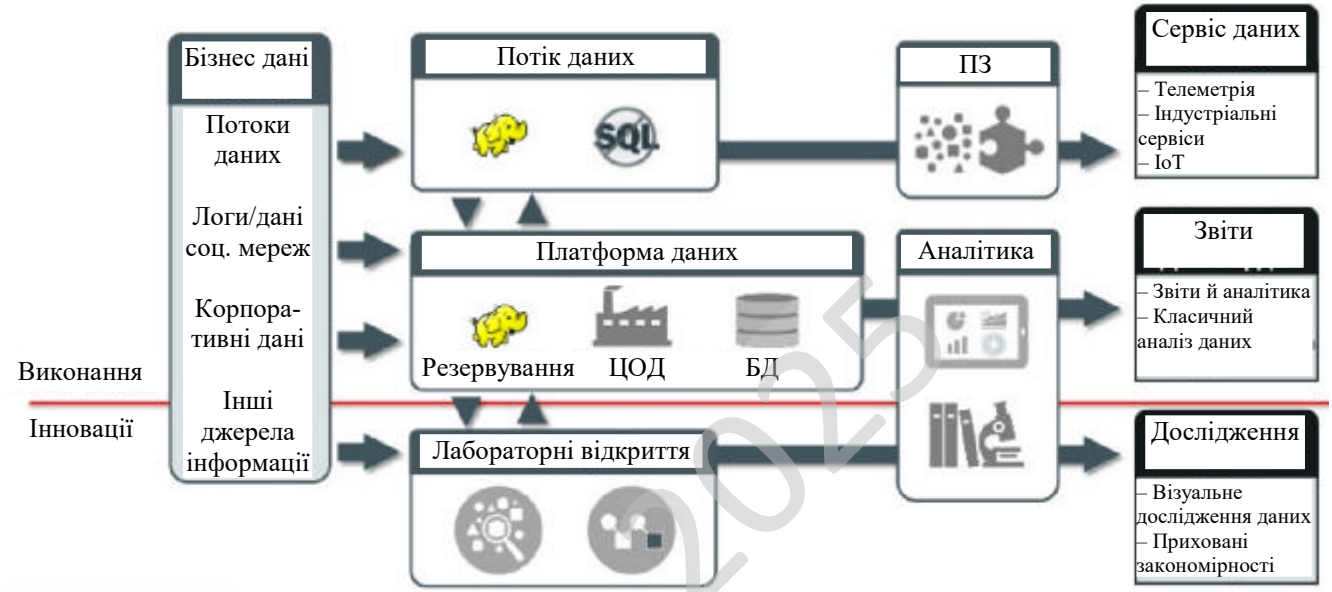

Рисунок 3.1 – Структурна схема системи

СУБД орієнтовану на IoT та Big Data можна розгорнути як на фізичних серверах, так і в публічній або приватній хмарі. Уже підготовлені рекомендації з її розгортання в хмарах Amazon і Microsoft, накопичений досвід інтеграції практично з усіма популярними аналітичними платформами й додатками. До речі, у Україні вже є сертифіковані партнери з досвідом впровадження СУБД орієнтованої на IoT та Big Data, є партнер, що може взяти на себе навчання фахівців.

Керування СУБД виробляється за допомогою керуючої консолі, що входить у поставку й реалізована у вигляді веб-додатку.

Для обробки даних, що надходять у реальному часі, передбачений механізм, що дозволяє завантажувати дані відразу в оперативну пам'ять, але її

обсяг завжди обмежений. Обмеження вдалося зняти завдяки реалізованій цього року інтеграції з Kafka – продуктом з відкритим вихідним кодом, що представляє собою сервісну шину, що обробляє повідомлення й забезпечує двосторонній обмін даними із СУБД орієнтованою на IoT та Big Data, що дозволяє не тільки відслідковувати події Інтернету речей, але й реагувати на них, генеруючи керуючого впливу.

Протягом найближчого років планується щорічно випускати один новий реліз і п'ять пакетів з істотними поліпшеннями (функціональних апдейтів). Замовники, які придбали послугу технічної підтримки СУБД орієнтованої на IoT та Big Data, стануть одержувати всі відновлення безкоштовно.

3.3 Розробка функціональної схеми

Функціонально СУБД орієнтована на IoT та Big Data, яка розробляється у даній роботі, призначена для обробки результатів наукових досліджень. На рисунку 3.2 зображена функціональна схема розробленої системи.

Вона складається з наступних функціональних блоків, які взаємодіють між собою, та призначених для виконання наступних дій:

– Комплекс автоматизації експериментальних установок призначений для автоматизації робіт на дослідницьких, технологічних і контрольно-діагностичних комплексах і експериментальних установках різного призначення.

– Комплекс може функціонувати на PC сумісному комп'ютері промислового, мобільного або офісного виконання, оснащеного засобами збору даних.

– Можливості комплексу по кількості, составу й характеристикам вимірювальних каналів залежать від використаних пристроїв збору даних і потужності застосовуваного комп'ютера.

Рисунок 3.2 – Функціональна схема системи

Основу комплексу становить інтегроване середовище, що дозволяє:

– проводити налаштування експерименту;

– здійснювати пошук потрібного сценарію в базі даних;

– робити запуск програмного забезпечення експерименту реального часу;

– здійснювати перегляд і аналіз результатів.

Програмне забезпечення експерименту реального часу складається із двох незалежних частин здатних працювати як єдине ціле на одному комп'ютері або поодинці на окремих комп'ютерах.

Підсистема тарировки й інформаційного супроводу вимірювальних

каналів передає інформацію іншим підсистемам комплексу про склад і характеристики наявних вимірювальних каналів і їхніх елементів, дозволяє формувати вимірювальні канали й визначати їхні метрологічні характеристики.

Комплекс може функціонувати як на одиночному комп'ютері, так і з використанням клієнт-серверних технологій у рамках розподіленої системи збору й обробки даних.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється. Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Рисунок 3.3 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схеми є основою ПЗ. Тому від точності і детальності проробки блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації, також те, що при розробці програми слід надати особливу увагу модулю СУБД орієнтованої на IoT та Big Data.

Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні блоки можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірки поточного стану та поверненням на початок схеми чи з завершенням роботи розробленого ПЗ.

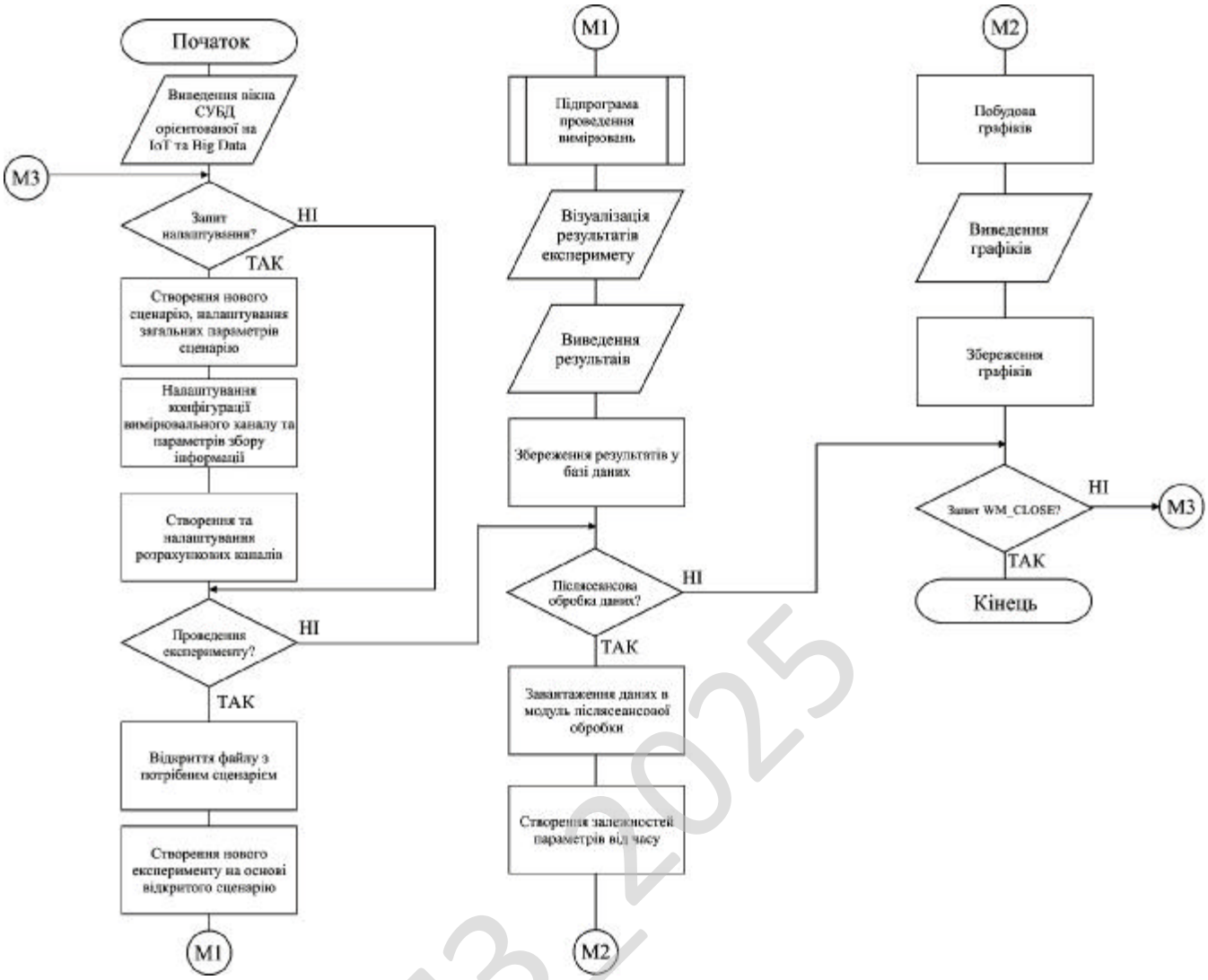

Рисунок 4.1 – Блок-схема основної програми

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

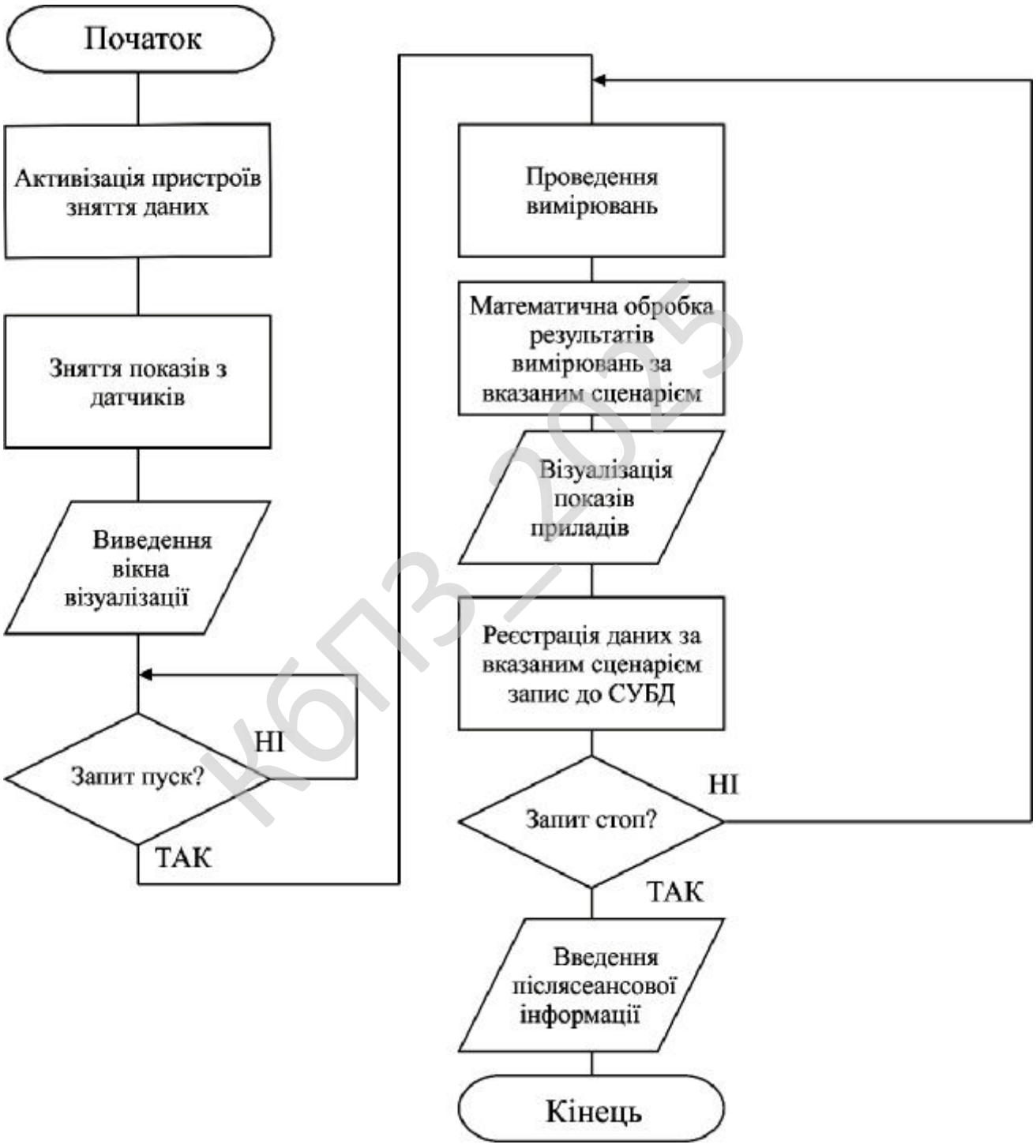

Рисунок 4.2 – Блок-схема роботи підпрограми

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Розглянемо використані технології та їх основні компоненти що підтверджують правильність використаних проектних рішень.

Redmine – вільне серверне ПЗ для управління проектами та відстежування помилок. До системи входить календар-планувальник та діаграми Ганта для візуального представлення ходу робіт за проектом та строків виконання. Redmine написано на мові Ruby і є ПЗ розробленим з використанням відомого веб-фреймворку Ruby on Rails, що означає легкість в розгортанні системи та її адаптації під конкретні вимоги. Для кожного проекту можна вести свої вікі та форуми.

Функціональні можливості:

– Ведення декількох проектів.

– Гнучка система доступу з використанням ролей.

– Система відстеження помилок.

– Діаграми Ганта та календар.

– Ведення новин проекту, документів та управління файлами.

– Сповіщення про зміни за допомогою RSS-потоків та електронної пошти.

– Власна Wiki для кожного проекту.

– Форуми для кожного проекту.

– Облік часових витрат.

– Налаштування власних (custom) полів для задач, затрат часу, проектів та користувачів.

– Легка інтеграція із системами керування версіями (SVN, CVS, Git, Mercurial, Bazaar и Darcs).

– Створення записів про помилки на основі отриманих листів

– Підтримка LDAP автентифікації.

– Можливість самореєстрації нових користувачів.

– Багатомовний інтерфейс (у тому числі українська мова).

– Підтримка СКБД: MySQL, PostgreSQL, SQLite.

Діаграма Ганта (Gantt chart, також стрічкова діаграма, графік Ганта) – це популярний тип діаграм, який використовується для ілюстрації плану, графіка робіт за будь-яким проектом. Є одним з методів планування та управління проектами.

Діаграма Ганта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі. Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і тривалості завдання. На деяких діаграмах Ганта також показується залежність між завданнями.

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню, заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні».

Часто діаграма Ганта використовується спільно з таблицею зі списком робіт, рядки якої відповідають окремо взятій задачі, зображеній на діаграмі, а стовпці містять додаткову інформацію про задачу.

Система відстеження помилок Багтрекер – прикладна програма для допомоги розробникам програмного забезпечення (програмістам, тестувальникам тощо) враховувати і контролювати помилки, знайдені у програмах, питання щодо функціональності, рішення та оновлення, побажання користувачів, а також стежити за процесом їх виконання.

Кожному, хто розробляв програмні продукти, добре знайоме співвідношення «20/80» – останні 20 % роботи тривають 80 % часу.

Як це не парадоксально, але нічого дивного в цій пропорції немає, адже саме на завершальній стадії починається тестування проекту, коли виявляються помилки, і що більший проект, то більше буде знайдено помилок.

Водночас досить часто виявляється, що більшість цих помилок були відомі та могли бути виправлені з меншими витратами на попередніх стадіях роботи, але не були вчасно описані, а потім загубилися серед інших важливих завдань.

Отже, система відстеження помилок у найпростішому варіанті – це процес, що включає в себе виявлення помилки, її опис, виправлення і перевірку цього виправлення, тобто процес «стеження» за багом протягом всього як його життєвого циклу, так і життєвого циклу розробки в цілому.

Сукупність інформації про дефект. Головний компонент такої системи – база даних, що містить відомості про виявлені дефекти. Ці відомості можуть включати в себе:

– номер (ідентифікатор) дефекту;

– хто повідомив про дефект;

– дата і час виявлення дефекту;

– версія продукту, в якій виявлено дефект;

– серйозність (критичність) дефекту та пріоритет рішення;

– опис кроків для відтворення дефекту (неправильної поведінки програми);

– відповідальний за усунення дефекту;

– обговорення можливих рішень та їх наслідків;

– поточний стан виправлення дефекту;

– версії продукту, в якій дефект виправлений.

Крім того, розвинені системи надають можливість прикріплювати файли, які допомагають описати проблему, наприклад, дамп пам'яті або скріншот.

Використання. Основна перевага систем відстеження помилок полягає в забезпеченні чітких централізованих оглядів, запитів на розробку (включаючи помилки і виправлення) та їх стан. У корпоративному середовищі, системи відстеження помилок можуть бути використані для генерації звітів по

продуктивності програмістів виправлення помилок. Однак, це може іноді приводити до неточних результатів, тому що різні помилки можуть мати різні ступені пріоритету та серйозності, що пов'язано з складністю їх фіксації.

Життєвий цикл дефекту. Як правило, система відстеження помилок використовує той чи інший варіант «життєвого циклу» помилки, стадія якого визначається поточним станом помилки.

Типовий життєвий цикл дефекту:

1. Новий – дефект зареєстрований тестувальником.

2. Призначений – призначений відповідальний за виправлення дефекту.

3. Дозволений – дефект переходить назад у сферу відповідальності тестувальника. Як правило, супроводжується резолюцією, наприклад:

– Виправлено (виправлення включені у версію таку-то).

– Дубль (повторює дефект, що вже знаходиться в роботі).

– Не виправлено (працює відповідно до специфікації, має занадто низький пріоритет, виправлення відкладено до наступної версії тощо).

– «В мене все працює» (запит додаткової інформації про умови, в яких дефект проявляється).

4. Далі тестувальник проводить перевірку виправлення, залежно від чого дефект або знову переходить у стан «Призначений» (якщо він описаний як виправлений, але не виправлений), або у стан «Закрито».

5. Відкрито повторно – дефект знайдено знову в іншій версії.

Система може надавати адміністраторові можливість налаштування користувачі, які можуть переглядати і редагувати помилки залежно від їх стану, переводити їх в інший стан або видаляти.

У корпоративному середовищі, система відстеження помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає достатньо точних результатів через те, що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми прямо не стосується

складності її усунення.

При розробці ПЗ було використано підходи ри́зик-ме́неджменту – це система управління ризиками, яка включає в себе стратегію та тактику управління, направлені на досягнення основних цілей. Ефективний ризик-менеджмент включає:

– систему управління;

– систему ідентифікації і вимірювання;

– систему супроводження (моніторингу та контролю).

Сучасна наука представляє ризик як вірогідну подію, в результаті настання якої можуть відбутися позитивні, нейтральні або негативні наслідки. Якщо ризик припускає наявність як позитивних, так і негативних результатів, він відноситься до спекулятивних ризиків. Якщо ж наслідки негативні, або відсутні взагалі, такий ризик іменується чистим.

Мета ризик-менеджменту – підвищення конкурентоспроможності господарюючих суб'єктів за допомогою захисту від реалізації чистих ризиків.

Теорія ризик-менеджменту ґрунтується на трьох базових поняттях: корисності, регресії і диверсифікації.

У 1738 швейцарський математик Даніель Бернуллі доповнив теорію вірогідності методом корисності або привабливості того або іншого результату подій. Ідея Бернуллі полягала в тому, що в процесі ухвалення рішення люди приділяють більше уваги розміру наслідків різних результатів, ніж їх вірогідність.

В кінці XIX століття англійський дослідник Ф. Гальтон запропонував вважати регресію або повернення до середнього значення універсальною статистичною закономірністю. Суть регресії трактувалася ним як повернення явищ до норми з часом. Згодом було доведено, що правило регресії діє в найрізноманітніших ситуаціях, починаючи з азартних ігор та розрахунку вірогідності виникнення нещасних випадків, і закінчуючи прогнозуванням коливань економічних циклів.

У 1952 аспірант Університету Чикаго Гарі Марковіц в статті «Диверсифікація вкладень» («Portfolio Selection») математично обґрунтував стратегію диверсифікації інвестиційного портфеля, зокрема, він показав, як шляхом продуманого розподілу вкладень мінімізувати відхилення прибутковості від очікуваного показника. У 1990 Г. Марковіцу присуджена Нобелівська премія за розробку теорії і практики оптимізації портфеля фондових активів.

Етапи ризик-менеджменту

У ризик-менеджменті прийнято виділяти декілька ключових етапів:

– на першому етапі відбувається виявлення ризику з супутньою оцінкою вірогідності його реалізації і масштабу наслідків;

– на другому етапі здійснюється розробка ризик-стратегії з метою зниження вірогідності реалізації ризику і мінімізації можливих негативних наслідків;

– на третьому етапі вибираються методи і інструменти управління виявленим ризиком;

– на четвертому етапі проводиться безпосереднє управління ризиком;

– на завершальному етапі оцінюються досягнуті результати і коректується ризик-стратегія.

За ключовий етап ризик-менеджменту вважається етап вибору методів і інструментів управління ризиком.

Методи і інструментарій ризик-менеджменту

Базовими методами ризик-менеджменту є відмова від ризиків, зниження, передача і ухвалення.

Ризик-інструментарій значно ширший. Він включає політичні, організаційні, правові, економічні, соціальні інструменти, причому ризик-менеджмент як система допускає можливість одночасного застосування декількох методів і інструментів ризик-управління.

Найбільш часто вживаним інструментом ризик-менеджменту є страхування. Страхування припускає передачу відповідальності за відшкодування передбачуваного збитку сторонній організації (страхової компанії).

Прикладами інших інструментів можуть бути відмова від надмірно ризикової діяльності (метод відмови), профілактика або диверсифікація (метод зниження), аутсорсинг витратних ризикових функцій (метод передачі), формування резервів або запасів (метод ухвалення).

Була використана водоспадна (каскадна) модель життєвого циклу ПЗ (waterfall model) – послідовний метод розробки програмного забезпечення, названий так через діаграму схожу на водоспад.

Ця модель розробки запозичена з системної інженерії у виробництві та будівництві – областях, в яких зміни на пізніх етапах дуже дорогі, або неможливі. Наприклад, для створення складних інженерних конструкцій (споруд, літаків, мостів і т.п.). Зміни в проекті фундаменту будинку після того, як покладений дах коштують дуже дорого, тому перфекціонізм на початкових етапах проектування просто необхідний. Інженери, які починали займатись розробкою програмного забезпечення перейшовши з інших галузей, просто адаптували звичну модель, тому що на ранніх етапах розвитку комп'ютерної техніки не було методологій створених саме для програмування. Проте, схожі методології застосовуються для програмного забезпечення й далі, у випадках коли вимоги фіксовані, і вимагається висока якість та надійність, наприклад в системах для військових чи медичних потреб.

Перший формальний опис водоспадної моделі, після якої вона стала популярною був здійснений В. В. Ройсом у 1970. Попри те, що стаття містить переважно критику методу, на неї часто посилаються.

Переваги методу:

– Ніяких переробок.

– Гарна специфікація перетікає в гарну документацію.

– Зрозуміла модель.

– Розробники можуть мати низьку кваліфікацію.

Недоліки:

– Необхідний перфекціонізм на кожному етапі.

– Важко вносити зміни (якщо взагалі можливо).

– Надлишкове проектування.

– Поділ розробників на "perfect" та "code monkeys".

Модифікації. Через те що цей метод погано підходить для розробки саме ПЗ, частіше використовують його модифікації.

Найвідоміша модифікація – Sashimi. Названа так через японську страву сашимі (суші нарізане і сервіроване так, що складені рядочком шматочки накладаються один на одного). В моделі розробки Сашимі фази життєвого циклу йдуть одна за одною, але при цьому перекриваються одна з одною в часі.

4.2 Захист розробленого програмного забезпечення

Дані у програмному забезпеченні я захищаю за допомогою NTRU. NTRUEncrypt (абревіатура Nth-degree TRUncated polynomial ring або Number Theorists aRe Us) – це криптографічна система з відкритим ключем, що раніше називалася NTRU.

Криптосистема NTRUEncrypt, заснована на ґратчастій криптосистемі, створена як альтернатива RSA і криптосистемам на еліптичних кривих (ECC). Стійкість алгоритму забезпечується труднощами пошуку найкоротшого вектора ґрати, що більше стійка до атак, здійснюваним на квантових комп'ютерах. На відміну від своїх конкурентів RSA,ECC, Elgamal, алгоритм використовує операції над кільцем:

$$\mathbb{Z}[X]/(X^N - 1),$$

усічених багаточленів ступеня, що не перевершує $N - 1$:

$$\mathbf{a}(X) = \mathbf{a} = a_0 + a_1 X + a_2 X^2 + \cdots + a_{N-2} X^{N-2} + a_{N-1} X^{N-1}.$$

Такий багаточлен можна також представити вектором:

$$\vec{a}(X) = \vec{a} = \sum_{i=0}^{N-1} a_i X^i = [a_0, a_1, a_2, \cdots, a_{N-2}, a_{N-1}]$$

Як і будь-який молодий алгоритм, NTRUEncrypt погано вивчений, хоча й був офіційно затверджений для використання в сфері фінансів комітетом Accredited Standards Committee X9.[1]

Існує реалізації NTRUEncrypt з відкритим вихідним кодом.[2]

NTRUEncrypt, що споконвічно називався NTRU, був винайдений в 1996 році й представлений на конференціях CRYPTO, Конференція RSA, Eurocrypt. Причиною, що послужила початком розробки алгоритму в 1994 році, стала стаття [3], у якій говорилося про легкість злому існуючих алгоритмів на квантових комп'ютерах, які, як показало час, не за горами[4]. У цьому ж році, математики Jeffrey Hoffstein, Jill Pipher і Joseph H. Silverman, що розробили систему разом із засновником компанії NTRU Cryptosystems, Inc. (пізніше перейменованої в SecurityInnovation), Даніелєм Лієманом (Daniel Lieman) запатентували свій винахід.[5]

Кільця усічених багаточленів

NTRU оперує над багаточленами ступеня не переважаючої $N-1$:

$$\mathbf{a} = a_0 + a_1 X + a_2 X^2 + \cdots + a_{N-2} X^{N-2} + a_{N-1} X^{N-1},$$

де коефіцієнти $a_0, \cdots, a_{N-1}$ – цілі числа. Щодо операцій додавання й множення за модулем багаточлена $X^N - 1$.Такі багаточлени утворять кільце *R*, називане кільцем усічених багаточленів, що ізоморфно кільцю відносин:

$$\mathbb{Z}[X]/(X^N - 1).$$

NTRU використовує кільце усічених багаточленів *R* разом з діленням за модулем на взаємно прості числа *p* і *q* для зменшення коефіцієнтів багаточленів.

У роботі алгоритму також використовуються зворотні багаточлени в кільці усічених багаточленів. Слід зазначити, що не всякий багаточлен має зворотний, але якщо зворотний поліном існує, то його легко обчислити.[6][7]

Генерація відкритого ключа

Для передачі повідомлення від Аліси до Боба необхідні відкритий і закритий ключі. Відкритий знають як Боб, так і Аліса, закритий ключ знає тільки Боб, що він використовує для генерації відкритого ключа. Для цього Боб вибирає два «маленьких» поліноми *f g* з *R*. «Малість» поліномів мається на увазі в тому розумінні, що він маленький щодо довільного полінома за модулем *q*: у довільному поліномі коефіцієнти повинні бути приблизно рівномірно розподілені за модулем *q*, а в малому поліномі вони багато менше *q*. Малість поліномів визначається за допомогою чисел *df* і *dg*:

Поліном *f* має *df* коефіцієнтів рівних «1» і *df*-1коефіцієнтів рівних «-1», а інші – «0». У цьому випадку говорять, що:

$$\mathbf{f} \in \mathcal{L}_f$$

Поліном *g* має *dg* коефіцієнтів рівних «1» і стільки ж рівних «-1», інші – «0» У цьому випадку говорять, що:

$$\mathbf{g} \in \mathcal{L}_g$$

Причина, по якій поліноми вибираються саме таким чином, полягає в тому, що *f* , можливо, буде мати зворотний, а *g* – однозначно немає (*g* (1) = 0, а нульовий елемент не має зворотного).

Боб повинен зберігати ці полінома в секреті. Далі Боб обчислює зворотні поліноми $\mathbf{f}_p$ й $\mathbf{f}_q$, тобто такі, що:

$$\mathbf{f} \cdot \mathbf{f}_p \equiv 1 \pmod{p} \text{ й } \mathbf{f} \cdot \mathbf{f}_q \equiv 1 \pmod{q}$$

Якщо *f* не має зворотного полінома, то Боб вибирає інший поліном *f*.

Секретний ключ – це пари $(\mathbf{f}, \mathbf{f}_p)$, а відкритий ключ *h* обчислюється за формулою:

$$\mathbf{h} = (p\mathbf{f}_q \cdot \mathbf{g}) \bmod q.$$

Приклад:

Для приклада візьмемо *df*=4, а *dg*=3. Тоді як поліноми можна вибрати

$$\mathbf{f} = -1 + X + X^2 - X^4 + X^6 + X^9 - X^{10}$$
$$\mathbf{g} = -1 + X^2 + X^3 + X^5 - X^8 - X^{10}$$

Далі для полінома *f* шукаються зворотні поліноми за модулем *p*=3 иq=32:

$$\mathbf{f}_p = 1 + 2X + 2X^3 + 2X^4 + X^5 + 2X^7 + X^8 + 2X^9$$

$$\mathbf{f}_q = 5+9X+6X^2+16X^3+4X^4+15X^5+16X^6+22X^7+20X^8+18X^9+30X^{10}$$

Заключним етапом є обчислення самого відкритого ключа *h*:

$$\mathbf{h} = (p\mathbf{f}_q \cdot \mathbf{g}) \bmod 32 = 8+25X+22X^2+20X^3+12X^4+24X^5+15X^6+19X^7+12X^8+19X^9+16X^{10}.$$

Шифрування

Тепер, коли в Аліси є відкритий ключ, вона може відправити зашифроване повідомлення Бобові. Для цього потрібно повідомлення представити у вигляді полінома m з коефіцієнтами за модулем *p*, обраними з діапазону (-*p*/2, *p*/2]. Тобто *m* є «малим» поліномом за модулем *q*. Далі Алісі необхідно вибрати інший «малий» поліном r, що називається «сліпучої», обумовлений за допомогою числа *dr*:

Поліном r має *dr* коефіцієнтів рівних «1» і стільки ж рівних «-1», інші – «0». У цьому випадку говорять, що:

$$\mathbf{r} \in \mathcal{L}_r.$$

Використовуючи ці поліноми, зашифроване повідомлення виходить за формулою:

$$\mathbf{e} = (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) \bmod q.$$

При цьому кожній, хто знає (або може обчислити) осліплюючий поліном r, зможе прочитати повідомлення *m*.

Приклад:

Припустимо, що Аліса хоче послати повідомлення, представлене у вигляді полінома:

$$\mathbf{m} = -1 + X^3 - X^4 - X^8 + X^9 + X^{10},$$

і вибрала «осліплюючий» поліном, для якого *dr*=3:

$$\mathbf{r} = -1 + X^2 + X^3 + X^4 - X^5 - X^7.$$

Тоді шифротекст *e*, готовий для передачі Бобові буде:

$$\mathbf{e} = (\mathbf{r}\cdot\mathbf{h}+\mathbf{m}) \bmod 32 = 14+11X+26X^2+24X^3+14X^4+16X^5+30X^6+7X^7+25X^8+6X^9+19X^{10}$$

Розшифрування

Тепер, одержавши зашифроване повідомлення *e*, Боб може його розшифрувати, використовуючи свій секретний ключ. Спочатку він одержує новий проміжний поліном:

$$\mathbf{a} = (\mathbf{f} \cdot \mathbf{e}) \bmod q.$$

Якщо розписати шифротекст, то одержимо ланцюжок:

$$\mathbf{a} = (\mathbf{f} \cdot \mathbf{e}) \bmod q = (\mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m})) \bmod q = (\mathbf{f} \cdot (\mathbf{r} \cdot p\mathbf{f}_q \cdot \mathbf{g} + \mathbf{m})) \bmod q$$

і остаточно:

$$\mathbf{a} = (p\mathbf{r} \cdot \mathbf{g} + \mathbf{f} \cdot \mathbf{m}) \bmod q.$$

Після того, як Боб обчислив поліном a за модулем *q*, він повинен вибрати його коефіцієнти з діапазону (-*q*/2, *q*/2] і далі обчислити поліном b, одержуваний з полінома a приведенням за модулем *p*:

$$\mathbf{b} = \mathbf{a} \bmod p = (\mathbf{f} \cdot \mathbf{m}) \bmod p,$$

так як:

$$(p\mathbf{r} \cdot \mathbf{g}) \bmod p = 0.$$

Тепер, використовуючи другу половину секретного ключа й отриманий поліном *b*, Боб може розшифрувати повідомлення:

$$\mathbf{c} = (\mathbf{f}_p \cdot \mathbf{b}) \bmod p.$$

Неважко бачити, що:

$$\mathbf{c} \equiv \mathbf{f}_p \cdot \mathbf{f} \cdot \mathbf{m} \equiv \mathbf{m} \pmod p.$$

У такий спосіб отриманий поліном *c* дійсно є вихідним повідомленням *m*.

Приклад:

Боб одержав від Аліси шифроване повідомлення *e*:

$$\mathbf{e} = 14+11X+26X^2+24X^3+14X^4+16X^5+30X^6+7X^7+25X^8+6X^9+19X^{10}$$

Використовуючи секретний ключ *f* Боб одержує поліном *a*:

$$\mathbf{a} = \mathbf{f}\cdot\mathbf{e} \pmod{32} = 3-7X-10X^2-11X^3+10X^4+7X^5+6X^6+7X^7+5X^8-3X^9-7X^{10} \pmod{32},$$

з коефіцієнтами, що належать проміжку (-*q*/2, *q*/2]. Далі перетворить поліном *a* у поліном *b*, зменшуючи коефіцієнти за модулем *p*.

$$\mathbf{b} = \mathbf{a} \pmod 3 = -X - X^2 + X^3 + X^4 + X^5 + X^7 - X^8 - X^{10} \pmod 3$$

Заключний крок – перемножування полінома *b* із другою половиною закритого ключа $\mathbf{f}_p$:

$$\mathbf{c} = \mathbf{f}_p \cdot \mathbf{b} = \mathbf{f}_p \cdot \mathbf{f} \cdot \mathbf{m} \pmod 3 = \mathbf{m} \pmod 3$$

$$\mathbf{c} = -1 + X^3 - X^4 - X^8 + X^9 + X^{10}$$

Який є вихідним повідомленням, що передавала Аліса.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ СУБД орієнтованої на IoT та Big Data яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Файл; IoT; BigData; Налаштування ПЗ; СУБД; Довідка.

– Розділу обрання Cloud системи зберігання БД.

– Розділу введення ключової послідовності.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

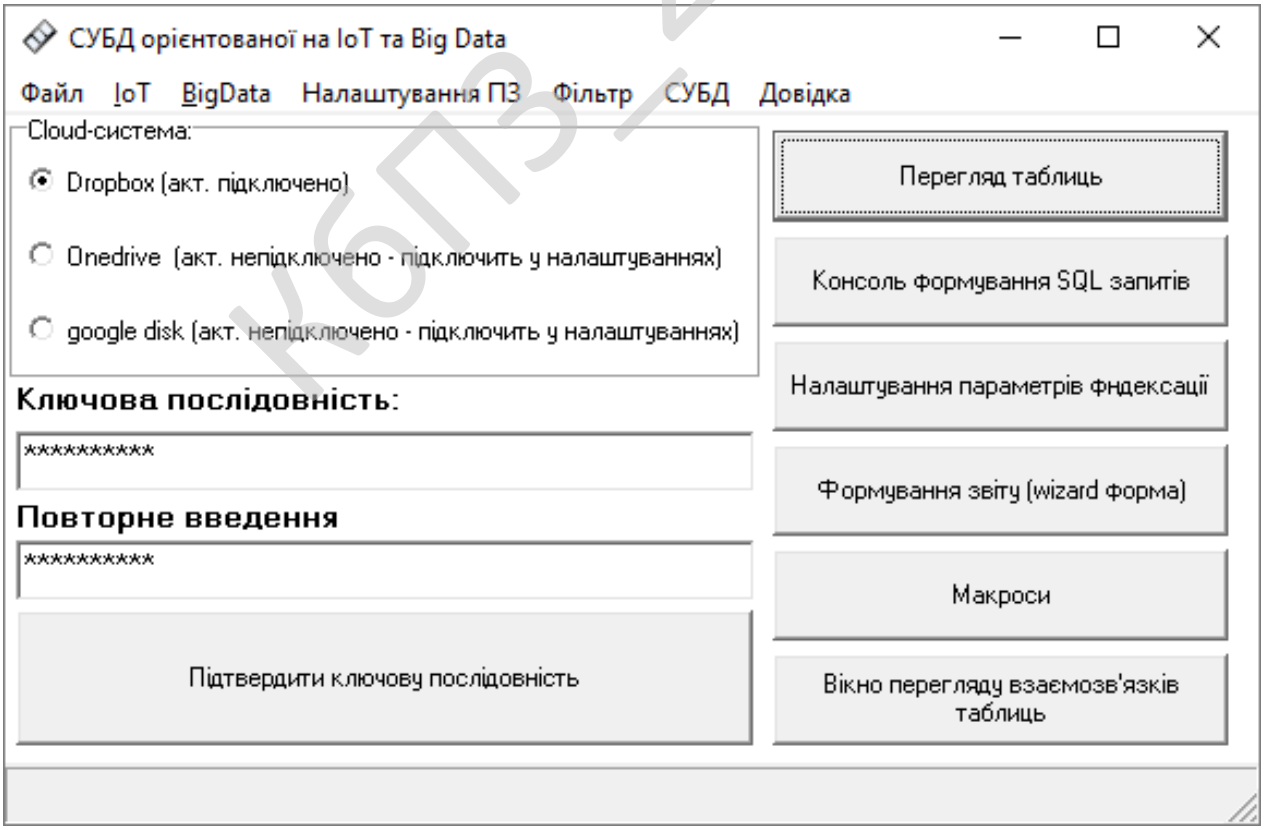

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і інтуїтивно зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

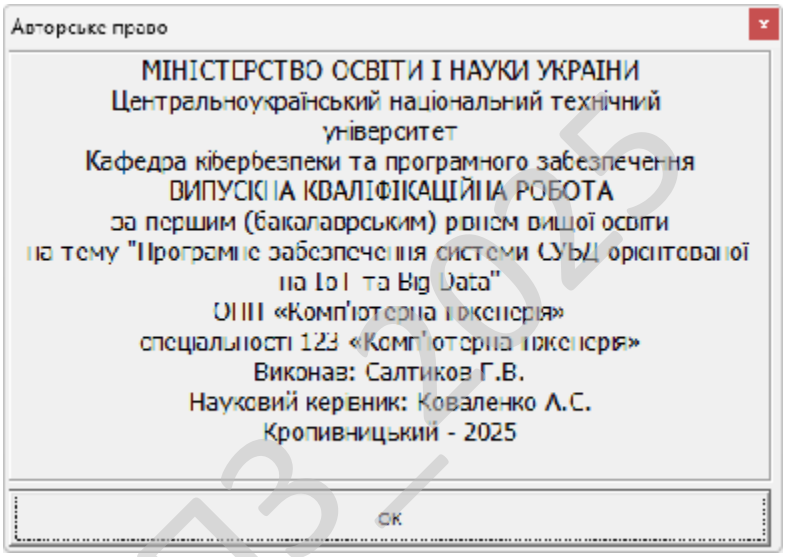

Рисунок 5.2 – Авторське право

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є

унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати IT рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене IT рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються

в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Обрано умови розповсюдження – Freeware.

Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів.

Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення.

Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи СУБД орієнтованої на IoT та Big Data.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем СУБД орієнтованої на IoT та Big Data.

– Досліджена система СУБД орієнтованої на IoT та Big Data.

– На основі отриманих результатів досліджень створена програмна реалізація системи СУБД орієнтованої на IoT та Big Data.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання СУБД орієнтованої на IoT та Big Data.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи СУБД орієнтованої на IoT та Big Data. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне

програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм NTRU.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Prateek Prasad. App Design Apprentice. Razeware LLC. 2020. 272 p.

2. Dawn Griffiths, David Griffiths. Head First Android Development. O'Reilly Media, Inc. 2021. 1414 p.

3. Nathan Metzler. Kotlin Programming for Beginners. Independently published. 2021. 158 p.

4. Aaron Torres. Go Programming Cookbook Second Edition. Packt Publishing Ltd. 2019. 427 p.

5. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.

6. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

7. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.

8. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.

9. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

10. Kuznetsov O., Ilchenko O., Kryvinska N., Buravchenko K., Smirnov O., Savchenko Iu. «An Empirical Assessment of Leading Blockchain Financial Services». *2023 IEEE 1st Ukrainian Distributed Ledger Technology Forum (UADLTF),* Kyiv, Ukraine, 2023, pp. 1-6,

11. Smirnov O., Fedorov E., Neskorodieva A., Neskorodieva T. «Intellectual Classification method of Gymnastic Elements Based on Combinations of

Descriptive and Generative Approache». *CEUR Workshop Proceedings* Volume 3664, 2024, Pages 11-23.

12. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

13. Malyukov V., Bebeshko B., Lakhno V., Smirnov O., Malyukova I., Mohylnyi H. «Managing the Purchase-Sale Process of Digital Currencies Under Fuzzy Conditions». *Lecture Notes in Networks and Systems,* 2023, 729 LNNS, pp. 104–112.

14. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

15. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

16. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

17. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

18. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

19. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143

20. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». Communications in Computer and Information Science, 2021, vol 1486. Springer, Cham. pp 169-184.

21. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

22. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

23. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

24. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». International Journal of Computer Network and Information Security (IJCNIS). Vol. 12, No. 3, 2020. PP.33-43.

25. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

26. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

27. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019,

Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

28. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

29. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

30. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P. 707-712.

31. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.

32. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

33. Вінтенко, Б., Миронець, І., Смірнов, О., Коваленко, А., Конопліцька-Слободенюк , О., Смірнова , Т., Константинова, Л. «Дослідження застосування систем підтримки оперативного персоналу об’єкту критичної інфраструктури при керуванні енергоблоком АЕС з реактором типу ВВЕР-1000».

Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 2024. № 2(26), С. 6-26.

34. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

35. Вінтенко Б.Ю., Смірнов О.А., Коваленко А.С., Смірнов С.А., Буравченко К.О. «Дослідження вимог міжнародних стандартів IEC60880 та IEC62138 з розробки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 3(73), С. 155-166.

36. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

37. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». Системи управління, навігації та зв'язку, 2023, вип. 2(72), С. 170-178.

38. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

39. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А. «Дослідження нормативної документації та стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». VI міжнародна науково-практична конференція "Інформаційна безпека та

комп'ютерні технології", м. Кропивницький. 20-21 квітня 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 35-36.

40. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

41. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

42. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

43. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

44. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

45. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

46. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

47. O. Smirnov, O. Kovalenko, A. Kovalenko, S. Smirnov, V. Vialkova. The mathematical model of the testing technology for DOM XSS vulnerabilities. Scientific & practical cyber security journal (SPCSJ) Vol 2 Issue 1, 22-28 pp. [Электронный Журнал]. Georgia. Tbilisi: SCSA – 2018.

48. Oleksii Smirnov, Oleksandr Kovalenko, Jamil Al-Azzeh, Anna Kovalenko, Serhii Smirnov. Qualitative risk analysis of software development. Asian Journal of Information Technology. – Volume 17(3). – Medwell Journals. – 2018. – P. 218-230.

49. Смірнов О.А., Коваленко О.В., Коваленко А.С., Смірнов С.А. Розробка методу передтестової компіляції й розподілу доступу. Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215

50. Smirnov Oleksii, Kovalenko Oleksandr, Kovalenko Anna, Smirnov Serhii. Method of testing the DOM XSS vulnerability. International Conference «Information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. 2017. P7.

51. Смірнов О.А., Смірнов С.А., Коваленко О.В., Коваленко А.С. Технологія тестування DOM XSS уразливості. Науково-практичний журнал кібер безпеки (SPCSJ) № 1. [Електронний журнал]. Грузія. Тбілісі: SCSA – 2017.

52. Смірнов О.А., Лисенко І.А. Інформаційна технологія проектування тестових наборів з урахуванням вимог до програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи СУБД орієнтованої на IoT та Big Data.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 47-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи СУБД орієнтованої на IoT та Big Data.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи СУБД орієнтованої на IoT та Big Data;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 63 аркуші.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 3.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Коваленко А.С.

Програмне забезпечення системи СУБД орієнтованої на IoT та Big Data

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 25

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
#Програмне забезпечення системи СУБД орієнтованої на IoT та Big Data
import random
import time
import datetime
import threading
import queue
import statistics
import json
import os

#Клас для представлення даних з IoT сенсорів
class SensorData:
    def __init__(self, device_id, timestamp, temperature, humidity, pressure):
        self.device_id = device_id
        self.timestamp = timestamp
        self.temperature = temperature
        self.humidity = humidity
        self.pressure = pressure

    def to_dict(self):
        return {
            "device_id": self.device_id,
            "timestamp": self.timestamp.isoformat(),
            "temperature": self.temperature,
            "humidity": self.humidity,
            "pressure": self.pressure
        }

    def __str__(self):
        return json.dumps(self.to_dict())

#Клас для симуляції IoT пристроїв, що надсилають дані
class IoTDevice(threading.Thread):
    def __init__(self, device_id, data_queue):
        threading.Thread.__init__(self)
        self.device_id = device_id
        self.data_queue = data_queue
        self.running = True

    def run(self):
        while self.running:
            current_time = datetime.datetime.now()
            temp = random.uniform(15.0, 35.0)
            hum = random.uniform(20.0, 80.0)
            pres = random.uniform(900.0, 1100.0)
            data = SensorData(self.device_id, current_time, temp, hum, pres)
            self.data_queue.put(data)
            #Засипання для симуляції інтервалу передачі даних
            time.sleep(random.uniform(0.5, 2.0))

    def stop(self):
        self.running = False

#Клас для збору даних від IoT пристроїв
class DataCollector:
    def __init__(self):
        self.data_queue = queue.Queue()
        self.devices = []
        self.collected_data = []
```

```
    def start_devices(self, count=5):
        for i in range(count):
            device = IoTDevice(f"Device-{i+1}", self.data_queue)
            device.start()
            self.devices.append(device)
            #Лог запуску пристрою
            print(f"Запущено пристрій Device-{i+1}")

    def stop_devices(self):
        for device in self.devices:
            device.stop()
        for device in self.devices:
            device.join()
        #Лог зупинки всіх пристроїв
        print("Всі пристрої зупинено.")

    def collect_data(self, duration=10):
        start_time = time.time()
        while time.time() - start_time < duration:
            try:
                data = self.data_queue.get(timeout=1)
                self.collected_data.append(data)
            except queue.Empty:
                continue
        #Лог завершення збору даних
        print(f"Зібрано {len(self.collected_data)} записів даних.")

#Клас, що імітує СУБД для зберігання та обробки даних
class BigDataDBMS:
    def __init__(self):
        self.tables = {}
        self.indexes = {}
        self.logs = []

    def create_table(self, table_name):
        if table_name not in self.tables:
            self.tables[table_name] = []
            self.logs.append(f"Таблицю {table_name} створено.")
            print(f"Таблицю {table_name} створено.")
        else:
            self.logs.append(f"Таблиця {table_name} вже існує.")
            print(f"Таблиця {table_name} вже існує.")

    def drop_table(self, table_name):
        if table_name in self.tables:
            del self.tables[table_name]
            self.logs.append(f"Таблицю {table_name} видалено.")
            print(f"Таблицю {table_name} видалено.")
        else:
            self.logs.append(f"Таблиця {table_name} не знайдена.")
            print(f"Таблиця {table_name} не знайдена.")

    def insert_record(self, table_name, record):
        if table_name in self.tables:
            self.tables[table_name].append(record)
            self.logs.append(f"Запис додано в таблицю {table_name}.")
        else:
            self.logs.append(f"Таблиця {table_name} не існує.")
            print(f"Таблиця {table_name} не існує.")

    def execute_query(self, table_name, query):
        results = []
        if table_name not in self.tables:
```

```
            self.logs.append(f"Таблиця {table_name} не знайдена для запиту.")
            print(f"Таблиця {table_name} не знайдена для запиту.")
            return results
        for record in self.tables[table_name]:
            if self.evaluate_record(record, query):
                results.append(record)
        self.logs.append(f"Запит виконано на таблиці {table_name}. Знайдено
{len(results)} записів.")
        return results

    def evaluate_record(self, record, query):
        for key, value in query.items():
            if not hasattr(record, key):
                return False
            if str(getattr(record, key)) != str(value):
                return False
        return True

    def update_record(self, table_name, query, update_values):
        if table_name not in self.tables:
            self.logs.append(f"Таблиця {table_name} не знайдена для оновлення.")
            print(f"Таблиця {table_name} не знайдена для оновлення.")
            return 0
        count = 0
        for record in self.tables[table_name]:
            if self.evaluate_record(record, query):
                for key, value in update_values.items():
                    if hasattr(record, key):
                        setattr(record, key, value)
                count += 1
        self.logs.append(f"Оновлено {count} записів у таблиці {table_name}.")
        return count

    def delete_record(self, table_name, query):
        if table_name not in self.tables:
            self.logs.append(f"Таблиця {table_name} не знайдена для видалення.")
            print(f"Таблиця {table_name} не знайдена для видалення.")
            return 0
        new_table = []
        count = 0
        for record in self.tables[table_name]:
            if self.evaluate_record(record, query):
                count += 1
            else:
                new_table.append(record)
        self.tables[table_name] = new_table
        self.logs.append(f"Видалено {count} записів з таблиці {table_name}.")
        return count

#Клас для обробки текстових запитів до СУБД
class QueryProcessor:
    def __init__(self, dbms):
        self.dbms = dbms

    def process(self, query_str):
        tokens = query_str.split()
        if len(tokens) < 4:
            print("Недійсний запит.")
            return None
        command = tokens[0].lower()
        if command == "select":
            table_name = tokens[tokens.index("from") + 1]
            query_conditions = {}
```

```
            if "where" in tokens:
                where_index = tokens.index("where")
                conditions_tokens = tokens[where_index + 1:]
                for cond in conditions_tokens:
                    if '=' in cond:
                        parts = cond.split('=')
                        if len(parts) == 2:
                            key = parts[0]
                            value = parts[1]
                            query_conditions[key] = value
            results = self.dbms.execute_query(table_name, query_conditions)
            return results
        elif command == "update":
            table_name = tokens[1]
            query_conditions = {}
            update_values = {}
            if "set" in tokens:
                set_index = tokens.index("set")
                if "where" in tokens:
                    where_index = tokens.index("where")
                    update_tokens = tokens[set_index+1:where_index]
                    conditions_tokens = tokens[where_index+1:]
                else:
                    update_tokens = tokens[set_index+1:]
                    conditions_tokens = []
                for upd in update_tokens:
                    if '=' in upd:
                        parts = upd.split('=')
                        if len(parts) == 2:
                            key = parts[0]
                            value = parts[1]
                            update_values[key] = value
                for cond in conditions_tokens:
                    if '=' in cond:
                        parts = cond.split('=')
                        if len(parts) == 2:
                            key = parts[0]
                            value = parts[1]
                            query_conditions[key] = value
                updated_count = self.dbms.update_record(table_name,
query_conditions, update_values)
                return updated_count
        elif command == "delete":
            table_name = tokens[tokens.index("from") + 1]
            query_conditions = {}
            if "where" in tokens:
                where_index = tokens.index("where")
                conditions_tokens = tokens[where_index+1:]
                for cond in conditions_tokens:
                    if '=' in cond:
                        parts = cond.split('=')
                        if len(parts) == 2:
                            key = parts[0]
                            value = parts[1]
                            query_conditions[key] = value
            deleted_count = self.dbms.delete_record(table_name,
query_conditions)
            return deleted_count
        else:
            print("Команда не підтримується.")
            return None

#Клас для виконання аналітичних операцій над даними
```

```
class AnalyticsEngine:
    def __init__(self, dbms):
        self.dbms = dbms

    def compute_average(self, table_name, field):
        if table_name not in self.dbms.tables:
            print(f"Таблиця {table_name} не знайдена для обчислення середнього
значення.")
            return None
        values = [getattr(record, field) for record in
self.dbms.tables[table_name] if hasattr(record, field)]
        if values:
            avg = statistics.mean(values)
            return avg
        return None

    def compute_max(self, table_name, field):
        if table_name not in self.dbms.tables:
            print(f"Таблиця {table_name} не знайдена для обчислення максимуму.")
            return None
        values = [getattr(record, field) for record in
self.dbms.tables[table_name] if hasattr(record, field)]
        if values:
            maximum = max(values)
            return maximum
        return None

    def compute_min(self, table_name, field):
        if table_name not in self.dbms.tables:
            print(f"Таблиця {table_name} не знайдена для обчислення мінімуму.")
            return None
        values = [getattr(record, field) for record in
self.dbms.tables[table_name] if hasattr(record, field)]
        if values:
            minimum = min(values)
            return minimum
        return None

    def compute_standard_deviation(self, table_name, field):
        if table_name not in self.dbms.tables:
            print(f"Таблиця {table_name} не знайдена для обчислення стандартного
відхилення.")
            return None
        values = [getattr(record, field) for record in
self.dbms.tables[table_name] if hasattr(record, field)]
        if values:
            std_dev = statistics.stdev(values)
            return std_dev
        return None

#Клас для резервного копіювання та відновлення даних
class BackupManager:
    def __init__(self, dbms, backup_folder="backups"):
        self.dbms = dbms
        self.backup_folder = backup_folder
        if not os.path.exists(self.backup_folder):
            os.makedirs(self.backup_folder)
            print("Створено папку для резервного копіювання.")

    def backup_table(self, table_name):
        if table_name not in self.dbms.tables:
            print(f"Таблиця {table_name} не існує для резервного копіювання.")
            return False
```

```
        backup_file = os.path.join(self.backup_folder,
f"{table_name}_backup.json")
        with open(backup_file, "w") as f:
            json_data = [record.to_dict() for record in
self.dbms.tables[table_name]]
            json.dump(json_data, f, indent=4)
        print(f"Резервне копіювання таблиці {table_name} збережено в
{backup_file}.")
        return True

    def restore_table(self, table_name):
        backup_file = os.path.join(self.backup_folder,
f"{table_name}_backup.json")
        if not os.path.exists(backup_file):
            print(f"Файл резервного копіювання для таблиці {table_name} не
знайдено.")
            return False
        with open(backup_file, "r") as f:
            json_data = json.load(f)
        self.dbms.create_table(table_name)
        for record_dict in json_data:
            timestamp =
datetime.datetime.fromisoformat(record_dict["timestamp"])
            record = SensorData(record_dict["device_id"], timestamp,
record_dict["temperature"], record_dict["humidity"], record_dict["pressure"])
            self.dbms.insert_record(table_name, record)
        print(f"Таблиця {table_name} відновлена з резервного копіювання.")
        return True

#Функція для симуляції періодичного оновлення даних
def periodic_update(dbms, table_name, interval=5, run_time=20):
    start_time = time.time()
    while time.time() - start_time < run_time:
        time.sleep(interval)
        update_query = {"device_id": "Device-1"}
        update_values = {"temperature": random.uniform(10, 40)}
        updated = dbms.update_record(table_name, update_query, update_values)
        print(f"Періодичне оновлення: оновлено {updated} записів.")

#Функція для симуляції складних запитів
def complex_query_simulation(query_processor):
    result1 = query_processor.process("select * from sensor_data where
device_id=Device-1")
    print("Результат запиту 1:")
    for record in result1:
        print(record)
    result2 = query_processor.process("update sensor_data set humidity=55 where
device_id=Device-2")
    print(f"Результат запиту 2: оновлено {result2} записів.")
    result3 = query_processor.process("delete from sensor_data where
device_id=Device-3")
    print(f"Результат запиту 3: видалено {result3} записів.")

#Додаткові допоміжні функції для розширення функціональності системи
def log_system_status(dbms):
    print("Стан системи:")
    for log in dbms.logs:
        print(log)

def simulate_heavy_load(data_collector, duration=10):
    start_time = time.time()
    while time.time() - start_time < duration:
        try:
```

```
                _ = data_collector.data_queue.get(timeout=0.5)
            except queue.Empty:
                continue
    print("Симуляція високого навантаження завершена.")

def dummy_function_one():
    print("Виконання допоміжної функції один.")
    for i in range(5):
        print(f"Допоміжний процес 1 - крок {i+1}")
        time.sleep(0.1)

def dummy_function_two():
    print("Виконання допоміжної функції два.")
    for i in range(5):
        print(f"Допоміжний процес 2 - крок {i+1}")
        time.sleep(0.1)

def dummy_function_three():
    print("Виконання допоміжної функції три.")
    for i in range(5):
        print(f"Допоміжний процес 3 - крок {i+1}")
        time.sleep(0.1)

def dummy_function_four():
    print("Виконання допоміжної функції чотири.")
    for i in range(5):
        print(f"Допоміжний процес 4 - крок {i+1}")
        time.sleep(0.1)

def dummy_function_five():
    print("Виконання допоміжної функції п'ять.")
    for i in range(5):
        print(f"Допоміжний процес 5 - крок {i+1}")
        time.sleep(0.1)

#Основна функція додатку
def main():
    print("Запуск системи СУБД для IoT та Big Data.")
    collector = DataCollector()
    collector.start_devices(count=8)
    #Збір даних від пристроїв
    collector.collect_data(duration=10)
    collector.stop_devices()
    dbms = BigDataDBMS()
    dbms.create_table("sensor_data")
    #Вставка зібраних даних у таблицю sensor_data
    for data in collector.collected_data:
        dbms.insert_record("sensor_data", data)
    qp = QueryProcessor(dbms)
    ae = AnalyticsEngine(dbms)
    #Виконання симуляції складних запитів
    complex_query_simulation(qp)
    #Резервне копіювання даних
    backup_mgr = BackupManager(dbms)
    backup_mgr.backup_table("sensor_data")
    #Виконання періодичного оновлення даних у окремому потоці
    update_thread = threading.Thread(target=periodic_update, args=(dbms,
"sensor_data", 3, 15))
    update_thread.start()
    #Виконання допоміжних функцій для розвантаження системи
    dummy_function_one()
    dummy_function_two()
    dummy_function_three()
```

```
    dummy_function_four()
    dummy_function_five()
    #Очікування завершення періодичного оновлення
    update_thread.join()
    #Виконання аналітичних операцій
    avg_temp = ae.compute_average("sensor_data", "temperature")
    max_temp = ae.compute_max("sensor_data", "temperature")
    min_temp = ae.compute_min("sensor_data", "temperature")
    std_temp = ae.compute_standard_deviation("sensor_data", "temperature")
    print(f"Середня температура: {avg_temp}")
    print(f"Максимальна температура: {max_temp}")
    print(f"Мінімальна температура: {min_temp}")
    print(f"Стандартне відхилення температури: {std_temp}")
    #Логування стану системи
    log_system_status(dbms)
    #Відновлення даних з резервного копіювання
    backup_mgr.restore_table("sensor_data")
    print("Завершення роботи системи СУБД.")

if __name__ == "__main__":
    main()

#Функція для додаткового тестування модулів системи
def additional_tests():
    print("Запуск додаткових тестів системи.")
    collector = DataCollector()
    collector.start_devices(count=3)
    collector.collect_data(duration=5)
    collector.stop_devices()
    dbms = BigDataDBMS()
    dbms.create_table("sensor_data")
    for data in collector.collected_data:
        dbms.insert_record("sensor_data", data)
    qp = QueryProcessor(dbms)
    result = qp.process("select * from sensor_data where device_id=Device-1")
    print("Результати додаткового тестування:")
    for record in result:
        print(record)
    print("Додаткові тести завершено.")

#Виклик додаткових тестів
additional_tests()

#Функція для симуляції масового завантаження даних
def mass_data_simulation():
    print("Початок симуляції масового завантаження даних.")
    collector = DataCollector()
    collector.start_devices(count=10)
    collector.collect_data(duration=15)
    collector.stop_devices()
    dbms = BigDataDBMS()
    dbms.create_table("sensor_data")
    for data in collector.collected_data:
        dbms.insert_record("sensor_data", data)
    print(f"Масове завантаження завершено, зібрано
{len(collector.collected_data)} записів.")
    qp = QueryProcessor(dbms)
    result = qp.process("select * from sensor_data where device_id=Device-5")
    print("Результати запиту для Device-5:")
    for record in result:
        print(record)
    print("Симуляція масового завантаження даних завершена.")
```

```
#Виклик функції симуляції масового завантаження даних
mass_data_simulation()

#Додаткові функції для розширення логіки системи
def auxiliary_function_a():
    print("Виконується допоміжна функція A.")
    for _ in range(3):
        time.sleep(0.2)
        print("Допоміжна функція A працює.")

def auxiliary_function_b():
    print("Виконується допоміжна функція B.")
    for _ in range(3):
        time.sleep(0.2)
        print("Допоміжна функція B працює.")

def auxiliary_function_c():
    print("Виконується допоміжна функція C.")
    for _ in range(3):
        time.sleep(0.2)
        print("Допоміжна функція C працює.")

def auxiliary_function_d():
    print("Виконується допоміжна функція D.")
    for _ in range(3):
        time.sleep(0.2)
        print("Допоміжна функція D працює.")

def auxiliary_function_e():
    print("Виконується допоміжна функція E.")
    for _ in range(3):
        time.sleep(0.2)
        print("Допоміжна функція E працює.")

#Виклик додаткових допоміжних функцій
auxiliary_function_a()
auxiliary_function_b()
auxiliary_function_c()
auxiliary_function_d()
auxiliary_function_e()

#Функція для тестування продуктивності системи
def performance_test():
    print("Початок тестування продуктивності системи.")
    collector = DataCollector()
    collector.start_devices(count=12)
    collector.collect_data(duration=10)
    collector.stop_devices()
    dbms = BigDataDBMS()
    dbms.create_table("sensor_data")
    for data in collector.collected_data:
        dbms.insert_record("sensor_data", data)
    start_perf = time.time()
    for _ in range(5):
        qp = QueryProcessor(dbms)
        _ = qp.process("select * from sensor_data where device_id=Device-2")
    end_perf = time.time()
    print(f"Тест продуктивності завершено. Час виконання: {end_perf -
start_perf} секунд.")

#Виклик тестування продуктивності
performance_test()
```

```
#Завершення коду
print("Всі операції виконано. Завершення роботи системи.")
```

Файл SensorDataStore.py

```
#!/usr/bin/env python3
#Імпорт необхідних бібліотек для роботи системи
import random
import time
import datetime
import threading
import asyncio
import json
import logging
import numpy as np
from flask import Flask, request, jsonify, abort
from functools import wraps
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from dask import delayed, compute

#Налаштування базового логування
logging.basicConfig(level=logging.DEBUG)
#Налаштування логування для детальної інформації
logger = logging.getLogger("IoT_BigData_System")
logger.setLevel(logging.DEBUG)

#=============================================================================
#Клас для зберігання даних сенсорів (імітація СУБД)
#=============================================================================
class SensorDataStore:
#Конструктор класу для ініціалізації сховища даних
    def __init__(self):
#Список для зберігання записів сенсорних даних
        self.data = []
#Блокування для потокобезпеки при доступі до даних
        self.lock = threading.Lock()
#Логування ініціалізації сховища даних
        logger.debug("Ініціалізовано сховище сенсорних даних.")

#Метод для додавання запису в сховище даних
    def add_record(self, record):
        with self.lock:
            self.data.append(record)
            logger.debug("Додано запис: %s", record)
#Затримка для симуляції операції запису
            time.sleep(0.05)

#Метод для отримання всіх записів
    def get_all_records(self):
        with self.lock:
            logger.debug("Отримання всіх записів з бази даних.")
            return list(self.data)

#Метод для отримання записів за ідентифікатором пристрою
    def get_records_by_device(self, device_id):
        with self.lock:
            filtered = [record for record in self.data if
record.get("device_id") == device_id]
            logger.debug("Отримано %d записів для пристрою %s", len(filtered),
device_id)
            return filtered

#Метод для отримання даних для машинного навчання
    def get_ml_data(self):
        with self.lock:
```

```
            X = []
            y = []
            for record in self.data:
                try:
                    #Використання вологості та тиску як ознак, температури як
мітки
                    X.append([float(record.get("humidity", 0)),
float(record.get("pressure", 0))])
                    y.append(float(record.get("temperature", 0)))
                except Exception as e:
                    logger.error("Помилка при обробці запису для ML: %s", e)
            logger.debug("Підготовлено дані для машинного навчання: %d записів",
len(y))
            return np.array(X), np.array(y)

#=========================================================================
#Модуль безпеки та аутентифікації для REST API
#=========================================================================
def token_required(f):
#Декоратор для перевірки токену автентифікації
    @wraps(f)
    def decorated(*args, **kwargs):
#Отримання токену з заголовків запиту
        token = request.headers.get("Authorization")
#Перевірка наявності токену
        if not token:
            logger.warning("Відсутній токен авторизації.")
            abort(401, description="Відсутній токен авторизації.")
#Перевірка відповідності токену з очікуваним значенням
        if token != "secret-token":
            logger.warning("Недійсний токен: %s", token)
            abort(403, description="Недійсний токен.")
#Логування успішної автентифікації
        logger.debug("Успішна автентифікація з токеном: %s", token)
        return f(*args, **kwargs)
    return decorated

#=========================================================================
#Модуль REST API для інтеграції системи
#=========================================================================
#Ініціалізація Flask додатку
app = Flask(__name__)
#Глобальне сховище сенсорних даних
sensor_store = SensorDataStore()

#Маршрут для отримання всіх записів сенсорних даних
@app.route("/api/data", methods=["GET"])
def api_get_all_data():
    logger.debug("Отримання всіх даних через API.")
    records = sensor_store.get_all_records()
    return jsonify(records)

#Маршрут для отримання записів за конкретним пристроєм
@app.route("/api/data/<device_id>", methods=["GET"])
def api_get_data_by_device(device_id):
    logger.debug("Отримання даних для пристрою: %s", device_id)
    records = sensor_store.get_records_by_device(device_id)
    return jsonify(records)

#Маршрут для додавання нового запису сенсорних даних (потребує автентифікації)
@app.route("/api/data", methods=["POST"])
@token_required
def api_add_data():
```

```
    if not request.json:
        logger.error("Невірний формат запиту: очікується JSON.")
        abort(400, description="Невірний формат запиту.")
    record = {
        "device_id": request.json.get("device_id"),
        "timestamp": datetime.datetime.now().isoformat(),
        "temperature": request.json.get("temperature"),
        "humidity": request.json.get("humidity"),
        "pressure": request.json.get("pressure")
    }
    sensor_store.add_record(record)
    logger.debug("Запис успішно додано через API: %s", record)
    return jsonify({"status": "Запис додано"}), 201

#Маршрут для отримання середньої температури (аналіз даних)
@app.route("/api/analytics/average", methods=["GET"])
def api_get_average_temperature():
    records = sensor_store.get_all_records()
    if not records:
        logger.warning("База даних порожня для обчислення середнього значення.")
        return jsonify({"average_temperature": None})
    temps = [record.get("temperature", 0) for record in records]
    avg_temp = sum(temps) / len(temps)
    logger.debug("Обчислено середню температуру: %f", avg_temp)
    return jsonify({"average_temperature": avg_temp})

#=============================================================================
#Модуль машинного навчання для аналізу сенсорних даних
#=============================================================================
class SensorMLModel:
#Конструктор для ініціалізації моделі машинного навчання
    def __init__(self):
        self.model = LinearRegression()
        self.trained = False
        logger.debug("Ініціалізовано модель машинного навчання.")

#Метод для тренування моделі з використанням даних з сенсорів
    def train_model(self, X, y):
        if len(X) == 0 or len(y) == 0:
            logger.error("Недостатньо даних для тренування моделі.")
            raise ValueError("Недостатньо даних для тренування моделі.")
        self.model.fit(X, y)
        self.trained = True
        logger.debug("Модель успішно натреновано з %d записів.", len(y))
        return "Модель натреновано успішно."

#Метод для передбачення температури з використанням моделі
    def predict(self, X):
        if not self.trained:
            logger.error("Модель не натренована. Неможливо виконати
передбачення.")
            raise ValueError("Модель не натренована.")
        predictions = self.model.predict(X)
        logger.debug("Передбачення виконано для %d записів.", len(predictions))
        return predictions.tolist()

#Метод для оцінки якості моделі з використанням MSE
    def evaluate_model(self, X, y):
        if not self.trained:
            logger.error("Модель не натренована для оцінки якості.")
            raise ValueError("Модель не натренована.")
        predictions = self.model.predict(X)
        mse = mean_squared_error(y, predictions)
```

```
        logger.debug("Обчислено MSE для моделі: %f", mse)
        return mse

#Глобальна змінна для зберігання екземпляра моделі машинного навчання
ml_model_instance = SensorMLModel()

#Маршрут для тренування моделі машинного навчання (потребує автентифікації)
@app.route("/api/ml/train", methods=["POST"])
@token_required
def api_ml_train():
    X, y = sensor_store.get_ml_data()
    try:
        result = ml_model_instance.train_model(X, y)
    except Exception as e:
        logger.error("Помилка під час тренування моделі: %s", e)
        abort(500, description="Помилка під час тренування моделі.")
    return jsonify({"status": result})

#Маршрут для виконання передбачення за допомогою натренованої моделі
@app.route("/api/ml/predict", methods=["POST"])
def api_ml_predict():
    if not request.json or "humidity" not in request.json or "pressure" not in
request.json:
        logger.error("Невірний формат запиту для передбачення.")
        abort(400, description="Невірний формат запиту для передбачення.")
    try:
        humidity = float(request.json.get("humidity"))
        pressure = float(request.json.get("pressure"))
    except Exception as e:
        logger.error("Невірні дані запиту: %s", e)
        abort(400, description="Невірні дані запиту.")
    X_new = np.array([[humidity, pressure]])
    try:
        predictions = ml_model_instance.predict(X_new)
    except Exception as e:
        logger.error("Помилка при виконанні передбачення: %s", e)
        abort(500, description="Помилка при виконанні передбачення.")
    return jsonify({"predicted_temperature": predictions[0]})

#=====================================================================
#Модуль розподіленої обробки даних з використанням Dask
#=====================================================================
def distributed_temperature_sum(sensor_data):
#Симуляція розподіленої обробки: обчислення суми температур за допомогою Dask
    tasks = []
    logger.debug("Створення завдань для розподіленої обробки температури.")
    for record in sensor_data:
#Створення відкладеної функції для кожного запису
        task = delayed(lambda temp: temp)(record.get("temperature", 0))
        tasks.append(task)
#Обчислення результатів завдань
    results = compute(*tasks)
    total = sum(results)
    logger.debug("Розподілена обробка завершена, загальна сума температур: %f",
total)
    return total

#Маршрут для отримання розподіленої суми температур
@app.route("/api/distributed/sum_temperature", methods=["GET"])
def api_distributed_sum_temperature():
    records = sensor_store.get_all_records()
    total = distributed_temperature_sum(records)
    return jsonify({"total_temperature": total})
```

```
#=========================================================================
#Модуль обробки подій з використанням asyncio
#=========================================================================
class EventProcessor:
#Конструктор для ініціалізації процесора подій
    def __init__(self):
#Створення нового асинхронного циклу подій
        self.loop = asyncio.new_event_loop()
#Ініціалізація асинхронної черги для подій
        self.queue = asyncio.Queue()
#Прапорець для контролю роботи процесора
        self.running = True
#Створення окремого потоку для виконання асинхронного циклу
        self.thread = threading.Thread(target=self.start_loop)
#Логування ініціалізації процесора подій
        logger.debug("Ініціалізовано процесор подій.")

#Метод для запуску асинхронного циклу у окремому потоці
    def start_loop(self):
        asyncio.set_event_loop(self.loop)
        logger.debug("Асинхронний цикл подій запущено.")
        self.loop.run_until_complete(self.process_events())

#Асинхронний метод для обробки подій із черги
    async def process_events(self):
        while self.running:
            try:
                event = await asyncio.wait_for(self.queue.get(), timeout=1.0)
                logger.debug("Обробка події: %s", event)
#Симуляція затримки під час обробки події
                await asyncio.sleep(0.5)
            except asyncio.TimeoutError:
                continue

#Метод для додавання нової події в чергу
    def post_event(self, event):
        if self.loop.is_running():
            self.loop.call_soon_threadsafe(self.queue.put_nowait, event)
            logger.debug("Подія додана: %s", event)
        else:
            logger.warning("Асинхронний цикл не працює. Неможливо додати подію:
%s", event)

#Метод для зупинки процесора подій
    def stop(self):
        self.running = False
        self.loop.call_soon_threadsafe(self.loop.stop)
        self.thread.join()
        logger.debug("Процесор подій зупинено.")

#Створення глобального екземпляра процесора подій
event_processor = EventProcessor()
#Запуск процесора подій у окремому потоці
event_processor.thread.start()

#=========================================================================
#Функція для симуляції надходження сенсорних даних
#=========================================================================
def simulate_sensor_data():
    while True:
#Генерація випадкових даних сенсора
        record = {
```

```
            "device_id": f"Device-{random.randint(1, 10)}",
            "timestamp": datetime.datetime.now().isoformat(),
            "temperature": round(random.uniform(15.0, 35.0), 2),
            "humidity": round(random.uniform(20.0, 80.0), 2),
            "pressure": round(random.uniform(900.0, 1100.0), 2)
        }
#Додавання запису в глобальне сховище
        sensor_store.add_record(record)
#Додавання події про надходження даних
        event_processor.post_event(f"Новий запис для {record['device_id']}")
#Логування симуляції надходження даних
        logger.debug("Симульовано новий запис: %s", record)
#Затримка для симуляції інтервалу надходження даних
        time.sleep(random.uniform(1.0, 3.0))

#Запуск симуляції надходження даних у фоновому потоці
sensor_thread = threading.Thread(target=simulate_sensor_data)
sensor_thread.daemon = True
sensor_thread.start()

#=============================================================================
#Основний блок для запуску Flask додатку
#=============================================================================
if __name__ == "__main__":
#Логування старту головного додатку
    logger.debug("Запуск головного додатку REST API.")
#Запуск Flask додатку на всіх інтерфейсах
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Файл DataVisualizer.py

```
#!/usr/bin/env python3
#============================================================================
# Модуль: Візуалізація даних
#============================================================================
import os
import time
import datetime
import threading
import random
import logging
import matplotlib.pyplot as plt
import numpy as np
import importlib.util
import functools

# Налаштування базового логування
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger("ExtendedModules")
logger.setLevel(logging.DEBUG)

#----------------------------------------------------------------------------
# Клас DataVisualizer
#----------------------------------------------------------------------------
class DataVisualizer:
#Конструктор для ініціалізації директорії збереження графіків
    def __init__(self, output_dir="visualizations"):
#Перевірка існування директорії, створення якщо відсутня
        self.output_dir = output_dir
        if not os.path.exists(self.output_dir):
            os.makedirs(self.output_dir)
            logger.debug("Створено директорію для візуалізацій: %s",
self.output_dir)

#Метод для побудови графіку залежності температури від часу
    def plot_temperature_over_time(self, records):
#Ініціалізація списків для часових міток та температур
        timestamps = []
        temperatures = []
        for record in records:
            try:
                ts = datetime.datetime.fromisoformat(record.get("timestamp"))
                temp = float(record.get("temperature"))
                timestamps.append(ts)
                temperatures.append(temp)
            except Exception as e:
                logger.error("Помилка обробки запису для графіку температури:
%s", e)
        if not timestamps:
            logger.warning("Немає даних для побудови графіку температури.")
            return
#Сортування даних за часовими мітками
        sorted_data = sorted(zip(timestamps, temperatures), key=lambda x: x[0])
        sorted_timestamps, sorted_temperatures = zip(*sorted_data)
        plt.figure(figsize=(10, 5))
        plt.plot(sorted_timestamps, sorted_temperatures, marker='o',
linestyle='-', color='blue')
        plt.title("Залежність температури від часу")
        plt.xlabel("Час")
        plt.ylabel("Температура")
        plt.grid(True)
#Збереження графіку у файл
```

```
        file_path = os.path.join(self.output_dir, "temperature_over_time.png")
        plt.savefig(file_path)
        plt.close()
        logger.debug("Графік температури збережено у файл: %s", file_path)

#Метод для побудови гістограми вологості
    def plot_humidity_histogram(self, records):
        humidities = []
        for record in records:
            try:
                humidity = float(record.get("humidity"))
                humidities.append(humidity)
            except Exception as e:
                logger.error("Помилка обробки даних вологості: %s", e)
        if not humidities:
            logger.warning("Немає даних для побудови гістограми вологості.")
            return
        plt.figure(figsize=(8, 6))
        plt.hist(humidities, bins=20, color='green', edgecolor='black')
        plt.title("Гістограма вологості")
        plt.xlabel("Вологість")
        plt.ylabel("Частота")
        file_path = os.path.join(self.output_dir, "humidity_histogram.png")
        plt.savefig(file_path)
        plt.close()
        logger.debug("Гістограма вологості збережена у файл: %s", file_path)

#Метод для побудови розсіянного графіку: тиск vs температура
    def plot_pressure_vs_temperature(self, records):
        pressures = []
        temperatures = []
        for record in records:
            try:
                pressure = float(record.get("pressure"))
                temperature = float(record.get("temperature"))
                pressures.append(pressure)
                temperatures.append(temperature)
            except Exception as e:
                logger.error("Помилка обробки даних для розсіянного графіку:
%s", e)
        if not pressures or not temperatures:
            logger.warning("Немає даних для побудови розсіянного графіку.")
            return
        plt.figure(figsize=(8, 6))
        plt.scatter(temperatures, pressures, color='red', alpha=0.6)
        plt.title("Залежність тиску від температури")
        plt.xlabel("Температура")
        plt.ylabel("Тиск")
        plt.grid(True)
        file_path = os.path.join(self.output_dir, "pressure_vs_temperature.png")
        plt.savefig(file_path)
        plt.close()
        logger.debug("Розсіянний графік збережено у файл: %s", file_path)

#=========================================================================
# Модуль: Система реального часу оповіщень
#=========================================================================
class NotificationSystem:
#Конструктор з налаштуваннями порогів сповіщень
    def __init__(self, temperature_threshold=30.0, humidity_threshold=70.0,
pressure_threshold=1050.0):
        self.temperature_threshold = temperature_threshold
        self.humidity_threshold = humidity_threshold
```

```
        self.pressure_threshold = pressure_threshold
        self.notification_log = []
        self.running = False
        self.thread = None
        logger.debug("Ініціалізовано систему сповіщень: temp=%f, humidity=%f,
pressure=%f",
                     self.temperature_threshold, self.humidity_threshold,
self.pressure_threshold)

#Метод перевірки запису та генерування сповіщень
    def check_and_notify(self, record):
        alerts = []
        try:
            temperature = float(record.get("temperature", 0))
            humidity = float(record.get("humidity", 0))
            pressure = float(record.get("pressure", 0))
            device_id = record.get("device_id", "Unknown")
            timestamp = record.get("timestamp",
datetime.datetime.now().isoformat())
            if temperature > self.temperature_threshold:
                alerts.append(f"Висока температура {temperature} на пристрої
{device_id}")
            if humidity > self.humidity_threshold:
                alerts.append(f"Висока вологість {humidity} на пристрої
{device_id}")
            if pressure > self.pressure_threshold:
                alerts.append(f"Високий тиск {pressure} на пристрої
{device_id}")
        except Exception as e:
            logger.error("Помилка перевірки даних для сповіщень: %s", e)
        for alert in alerts:
            self.send_notification(alert, timestamp)

#Метод для відправки сповіщення (імітація)
    def send_notification(self, message, timestamp):
        notification = {"timestamp": timestamp, "message": message}
        self.notification_log.append(notification)
        logger.info("Сповіщення: %s", notification)

#Функція моніторингу потоку даних та оповіщень
    def monitor_data_stream(self, data_source, interval=2.0):
        self.running = True
        while self.running:
            if data_source:
                record = data_source.pop(0)
                logger.debug("Моніторинг: обробка запису %s", record)
                self.check_and_notify(record)
            time.sleep(interval)

#Метод запуску моніторингу у окремому потоці
    def start_monitoring(self, data_source, interval=2.0):
        self.thread = threading.Thread(target=self.monitor_data_stream,
args=(data_source, interval))
        self.thread.start()
        logger.debug("Запущено моніторинг сповіщень.")

#Метод зупинки моніторингу
    def stop_monitoring(self):
        self.running = False
        if self.thread is not None:
            self.thread.join()
        logger.debug("Моніторинг сповіщень зупинено.")
```

```
#=========================================================================
# Модуль: Плагінова архітектура
#=========================================================================
#Інтерфейс для плагінів
class PluginInterface:
    def run(self, data):
        raise NotImplementedError("Метод run має бути реалізований плагіном.")

#Приклад плагіну A
class PluginA(PluginInterface):
    def run(self, data):
        logger.debug("Плагін A: обробка даних...")
        total_temp = sum([d.get("temperature", 0) for d in data])
        processed = {"plugin": "A", "total_temperature": total_temp}
        logger.debug("Плагін A: результат: %s", processed)
        return processed

#Приклад плагіну B
class PluginB(PluginInterface):
    def run(self, data):
        logger.debug("Плагін B: аналіз даних...")
        count = len(data)
        avg_humidity = sum([d.get("humidity", 0) for d in data]) / count if
count > 0 else 0
        processed = {"plugin": "B", "average_humidity": avg_humidity}
        logger.debug("Плагін B: результат: %s", processed)
        return processed

#Менеджер плагінів для завантаження та виконання
class PluginManager:
    def __init__(self, plugin_dir="plugins"):
        self.plugin_dir = plugin_dir
        self.plugins = {}
        logger.debug("Ініціалізовано менеджер плагінів з директорією: %s",
self.plugin_dir)

#Метод для завантаження плагінів з директорії
    def load_plugins(self):
        if not os.path.exists(self.plugin_dir):
            os.makedirs(self.plugin_dir)
            logger.debug("Створено директорію плагінів: %s", self.plugin_dir)
        for filename in os.listdir(self.plugin_dir):
            if filename.endswith(".py"):
                plugin_path = os.path.join(self.plugin_dir, filename)
                plugin_name = os.path.splitext(filename)[0]
                spec = importlib.util.spec_from_file_location(plugin_name,
plugin_path)
                if spec is not None:
                    module = importlib.util.module_from_spec(spec)
                    try:
                        spec.loader.exec_module(module)
                        for attr in dir(module):
                            attribute = getattr(module, attr)
                            if isinstance(attribute, type) and
issubclass(attribute, PluginInterface) and attribute is not PluginInterface:
                                self.plugins[plugin_name] = attribute()
                                logger.debug("Завантажено плагін: %s",
plugin_name)
                    except Exception as e:
                        logger.error("Помилка завантаження плагіну %s: %s",
plugin_name, e)

#Метод реєстрації плагіна вручну
```

```
    def register_plugin(self, name, plugin_instance):
        if isinstance(plugin_instance, PluginInterface):
            self.plugins[name] = plugin_instance
            logger.debug("Зареєстровано плагін: %s", name)
        else:
            logger.error("Невірний плагін: %s", name)

#Метод виконання всіх завантажених плагінів
    def execute_plugins(self, data):
        results = {}
        for name, plugin in self.plugins.items():
            try:
                logger.debug("Виконання плагіну: %s", name)
                result = plugin.run(data)
                results[name] = result
            except Exception as e:
                logger.error("Помилка виконання плагіну %s: %s", name, e)
        return results

#============================================================================
# Модуль: Кешування та оптимізація запитів
#============================================================================
class CacheManager:
    def __init__(self, expiration_time=60):
#Ініціалізація кешу як словника та встановлення терміну зберігання
        self.cache = {}
        self.expiration_time = expiration_time
        self.lock = threading.Lock()
        logger.debug("Ініціалізовано менеджер кешу з терміном: %d секунд",
self.expiration_time)

#Метод отримання результату з кешу за ключем
    def get_cached_result(self, key):
        with self.lock:
            entry = self.cache.get(key)
            if entry:
                result, timestamp = entry
                if (time.time() - timestamp) < self.expiration_time:
                    logger.debug("Знайдено кеш для ключа: %s", key)
                    return result
                else:
                    logger.debug("Кеш для ключа %s прострочено", key)
                    del self.cache[key]
            logger.debug("Кеш для ключа %s відсутній", key)
            return None

#Метод для встановлення значення у кеш
    def set_cached_result(self, key, result):
        with self.lock:
            self.cache[key] = (result, time.time())
            logger.debug("Встановлено кеш для ключа: %s", key)

#Декоратор для кешування результатів функцій
    def cache_decorator(self, func):
        @functools.wraps(func)
        def wrapper(*args, **kwargs):
            key = f"{func.__name__}:{args}:{kwargs}"
            cached = self.get_cached_result(key)
            if cached is not None:
                logger.debug("Повернення закешованого результату для функції:
%s", func.__name__)
                return cached
            result = func(*args, **kwargs)
```

```
            self.set_cached_result(key, result)
            return result
        return wrapper

# Створення глобального екземпляра менеджера кешу
cache_manager = CacheManager(expiration_time=120)

# Приклад складного запиту з використанням кешування
@cache_manager.cache_decorator
def complex_query(query):
    logger.debug("Виконання складного запиту: %s", query)
    time.sleep(2)
    result = {"query": query, "result": random.randint(1, 100)}
    logger.debug("Складний запит завершено: %s", result)
    return result

#=====================================================================
# Модуль: Очищення та перевірка даних
#=====================================================================
class DataCleaner:
    def __init__(self, temperature_range=(0, 50), humidity_range=(0, 100),
pressure_range=(800, 1200)):
        self.temperature_range = temperature_range
        self.humidity_range = humidity_range
        self.pressure_range = pressure_range
        logger.debug("Ініціалізовано модуль очищення з діапазонами:
температура=%s, вологість=%s, тиск=%s",
                     self.temperature_range, self.humidity_range,
self.pressure_range)

#Метод валідації одного запису
    def validate_record(self, record):
        errors = []
        try:
            temp = float(record.get("temperature", None))
            if temp is None or not (self.temperature_range[0] <= temp <=
self.temperature_range[1]):
                errors.append(f"Температура {temp} поза межами
{self.temperature_range}")
        except Exception as e:
            errors.append("Невірний формат температури")
        try:
            humidity = float(record.get("humidity", None))
            if humidity is None or not (self.humidity_range[0] <= humidity <=
self.humidity_range[1]):
                errors.append(f"Вологість {humidity} поза межами
{self.humidity_range}")
        except Exception as e:
            errors.append("Невірний формат вологості")
        try:
            pressure = float(record.get("pressure", None))
            if pressure is None or not (self.pressure_range[0] <= pressure <=
self.pressure_range[1]):
                errors.append(f"Тиск {pressure} поза межами
{self.pressure_range}")
        except Exception as e:
            errors.append("Невірний формат тиску")
        if errors:
            logger.warning("Помилки валідації запису: %s", errors)
        return errors

#Метод очищення одного запису
    def clean_record(self, record):
```

```
        cleaned = record.copy()
        try:
            cleaned["temperature"] = round(float(record.get("temperature", 0)),
2)
        except Exception as e:
            cleaned["temperature"] = None
            logger.error("Помилка перетворення температури: %s", e)
        try:
            cleaned["humidity"] = round(float(record.get("humidity", 0)), 2)
        except Exception as e:
            cleaned["humidity"] = None
            logger.error("Помилка перетворення вологості: %s", e)
        try:
            cleaned["pressure"] = round(float(record.get("pressure", 0)), 2)
        except Exception as e:
            cleaned["pressure"] = None
            logger.error("Помилка перетворення тиску: %s", e)
        if cleaned["temperature"] is None:
            cleaned["temperature"] = (self.temperature_range[0] +
self.temperature_range[1]) / 2
        if cleaned["humidity"] is None:
            cleaned["humidity"] = (self.humidity_range[0] +
self.humidity_range[1]) / 2
        if cleaned["pressure"] is None:
            cleaned["pressure"] = (self.pressure_range[0] +
self.pressure_range[1]) / 2
        logger.debug("Очищений запис: %s", cleaned)
        return cleaned

#Метод пакетного очищення списку записів
    def batch_clean(self, records):
        cleaned_records = []
        for record in records:
            errors = self.validate_record(record)
            if errors:
                logger.debug("Запис відфільтровано через помилки: %s", record)
                continue
            cleaned = self.clean_record(record)
            cleaned_records.append(cleaned)
        logger.debug("Пакетне очищення завершено: %d із %d записів",
len(cleaned_records), len(records))
        return cleaned_records

#============================================================================
# Основна функція для демонстрації роботи модулів
#============================================================================
if __name__ == "__main__":
#Створення симульованих даних сенсорів
    simulated_data = []
    for i in range(30):
        record = {
            "device_id": f"Device-{random.randint(1, 5)}",
            "timestamp": datetime.datetime.now().isoformat(),
            "temperature": random.uniform(10, 40),
            "humidity": random.uniform(30, 90),
            "pressure": random.uniform(850, 1150)
        }
        simulated_data.append(record)
        time.sleep(0.1)

# Використання модуля візуалізації даних
    visualizer = DataVisualizer()
    visualizer.plot_temperature_over_time(simulated_data)
```

```
    visualizer.plot_humidity_histogram(simulated_data)
    visualizer.plot_pressure_vs_temperature(simulated_data)

# Використання системи сповіщень для моніторингу даних
    notifier = NotificationSystem(temperature_threshold=35.0,
humidity_threshold=85.0, pressure_threshold=1100.0)
    data_for_monitoring = simulated_data.copy()
    notifier.start_monitoring(data_for_monitoring, interval=0.5)
    time.sleep(5)
    notifier.stop_monitoring()

# Використання менеджера плагінів
    plugin_manager = PluginManager()
    plugin_manager.register_plugin("PluginA", PluginA())
    plugin_manager.register_plugin("PluginB", PluginB())
    plugin_results = plugin_manager.execute_plugins(simulated_data)
    logger.debug("Результати плагінів: %s", plugin_results)

# Демонстрація роботи кешування запитів
    query_result_1 = complex_query("SELECT * FROM sensor_data WHERE temperature
> 25")
    query_result_2 = complex_query("SELECT * FROM sensor_data WHERE temperature
> 25")
    logger.debug("Перший результат запиту: %s", query_result_1)
    logger.debug("Другий результат запиту (з кешу): %s", query_result_2)

# Використання модуля очищення даних
    cleaner = DataCleaner(temperature_range=(5, 45), humidity_range=(10, 95),
pressure_range=(800, 1200))
    raw_data = simulated_data + [
        {"device_id": "Device-Error", "timestamp":
datetime.datetime.now().isoformat(), "temperature": "NaN", "humidity":
"unknown", "pressure": None},
        {"device_id": "Device-Error2", "timestamp":
datetime.datetime.now().isoformat(), "temperature": -10, "humidity": 150,
"pressure": 500}
    ]
    cleaned_data = cleaner.batch_clean(raw_data)
    logger.debug("Очищені дані: %s", cleaned_data)

    logger.info("Демонстрація роботи додаткових модулів завершена.")
```