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
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

за другим (магістерським) рівнем вищої освіти

на тему

“ Дослідження та програмна реалізація системи захисту в інформаційній системі”

Виконав здобувач вищої освіти
ІІ курсу, групи КН22М-1
ОПП «Комп'ютерні науки»
спеціальності 122 «Комп'ютерні науки»
_____ Мірошніков П.С.
«____» _____ 2023р.

Керівник проекту
кандидат технічних наук, доцент
_____ Пархоменко Ю.М.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *магістр*
Галузь знань *12 "Комп'ютерні науки"*
Спеціальність *122 "Комп'ютерні науки"*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерні науки"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
"____" _____20___року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Мірошнікову Павлу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи захисту і інформаційній системі*
2. Керівник роботи *Пархоменко Юрій Михайлович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 32-13 від 04.08.23
3. Строк подання роботи до захисту *20.12.2023 р.*
4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмне дослідження та програмна реалізація систем захисту ІС для IoT*
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. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Діаграма процесів процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	09.11.2023 р.	17.11.2023 р.
Охорона праці	Оришака О.В., к.т.н., доцент	03.11.2023 р.	21.11.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	12.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	18.10.2023 р.	
3.	Розробка моделі компонента	23.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	32.10.2023 р.	
6.	Програмування алгоритмів	11.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	16.11.2023 р.	
9.	Оформлення ПЗ	18.11.2023 р.	
10.	Попередній захист роботи	04.12.2023 р	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Мірошніков П.С. Дослідження та програмна реалізація системи захисту в інформаційній системі. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено та побудови серверних рішень які забезпечують роботу IoT з розробкою підсистеми безпеки передачі даних.

Метою розробки є дослідження та програмна реалізація для побудови системи серверних рішень, які забезпечують роботу Інтернету речей. Розробка хмарної платформи для Інтернету речей

Об'єктом дослідження є процес побудови рішень для робити IoT.

Предметом дослідження є серверні платформи для побудови рішень для роботи IoT.

Методи дослідження базуються на методах теорії кодування, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – розробка хмарної платформи для Інтернету речей.

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 архітектури IBM PC з ОС Windows XP/Vista/7/8/10/11.

Програму розроблено в середовищі Java та Scala.

Ключові слова: комп'ютерна науки, IoT, серверні платформи,обробка даних.

ABSTRACT

Miroshnikov P.S. Research and software implementation of a security system in an information system. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this master's thesis, software is developed, which is designed and the construction of server solutions that ensure the operation of IoT with the development of security subsystems for data transmission.

The purpose of the development is research and software implementation for building a system of server solutions that ensure the operation of the Internet of Things.

Development of a cloud platform for the Internet of Things

The object of research is the process of building solutions for making IoT.

The subject of research is server platforms for building solutions for IoT operation.

Research methods are based on coding theory methods, mathematical statistics methods, and software development methods.

The result of the work is the development of a cloud platform for the Internet of Things.

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 tools are provided.

The program can be used on PCs of IBM PC architecture with Windows XP/Vista/7/8/10/11.

The program was developed in the Java and Scala environment.

Keywords: computer science, IoT, server platforms, data processing.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ

AES (Advanced Encryption Standard)	симетричний шифр (стандарт).
DES (Data Encryption Standard)	симетричний шифр
PKI (Public Key Infrastructure)	інфраструктура відкритих ключів
IoT (Internet of Things)	концепція мережі, що складається із взаємозв'язаних фізичних пристроїв
SDL (Software Development Life Circle)	методика створення програмних продуктів
СКБД	система керування базами даних
HTTP	протокол передачі гіпертексту
ЕЦП	електронного цифрового підпису
ANSI/SPARC	американський національний інститут стандартів /стандарти і вимоги до планування комітету;- мова структурованих запитів
B2C (business-to-consumer)	("компанія-споживач)

ВСТУП

Актуальність теми. Інтернет речей є одним із найперспективніших напрямків розвитку інформаційно-комунікаційних технологій. Кількість пристроїв, підключених до Інтернету, стрімко зростає, а це вимагає сучасних підходів до побудови високонавантажених серверних систем. Платформи IoT повинні забезпечувати можливість аналізу різних аспектів даних, що необхідно для оптимізації різних виробничих та інших процесів. Проблеми і завдання в області побудови серверних систем для Інтернету речей можна розділити на загальні, які притаманні багатьом іншим системам обробки великих даних, і специфічні, які виникають тільки в цій сфері. Загальні цілі полягають у створенні та використанні ефективних, відмовостійких, масштабованих і розподілених систем даних.

Специфічними проблемами Інтернету речей є забезпечення надійного контролю та моніторингу пристроїв. У цьому документі визначаються вимоги до створення хмарних платформ IoT, досліджуються існуючі платформи та визначаються способи їх покращення. Досліджено й описано архітектурні рішення в області обробки великих обсягів даних і підходи до розробки програмного забезпечення, які дозволяють реалізувати платформу таким чином, щоб вона задовольняла заявленим функціональним і нефункціональним вимогам. Вивчаються можливості сторонніх рішень, які можна використовувати для створення платформи. Підтверджено вибір протоколу та формату даних, що дозволяє забезпечити найкращі параметри системи. Кінцевою метою роботи є побудова хмарної платформи для Інтернету речей, яка реалізує основні підсистеми, які необхідні таким системам, а саме: прийом, зберігання, обробка оперативних даних; аутентифікація та безпека; робота з адміністративними даними; моніторинг. Також не останнє місце займає питання інформаційної безпеки, адже доступ до такого будинку може завдати великої шкоди його власнику. Оскільки дистанційне управління, доступ до інформації і т.д. Досить

поширені в наші дні безпечні схеми, схеми шифрування та захисту повинні використовуватися, щоб мінімізувати вразливі місця та запобігти заподіянню шкоди злочинцям.

Мета й завдання дослідження. Метою роботи є дослідження та побудова серверних рішень які забезпечують роботу IoT з розробкою підсистеми безпеки передачі даних.

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

– огляд існуючих систем IoT з підсистемою безпеки передачі даних;

– дослідження та та побудова серверних рішень які забезпечують роботу IoT;

– програмна реалізація системи управління з підсистемою безпеки передачі даних.

Об'єктом дослідження є процес побудови серверного рішення хмарної платформи для IoT.

Предметом дослідження є методи реалізації побудови серверних рішень Хмарних систем.

Методи дослідження базуються на методах теорії кодування,методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– удосконалено серверне рішення хмарної платформи Інтернету речей з впровадженням підсистеми безпеки передачі даних;

– проведено огляд технологій зв'язку в системах IoT;

– розроблено вітчизняний продукт управління хмарною платформою з підсистемою безпеки передачі даних, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для реалізації систем серверних рішень в управлінні Хмарними платформами.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Апробація роботи відбулася на VII Міжнародній науково-практичній конференцію до 30-ти річчя кафедри кібербезпеки та програмного забезпечення «Інформаційна безпека та комп’ютерні технології» м.Кропивницький.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи серверного рішення Хмарної платформи для IoT з підсистемою безпеки передачі даних, є актуальною задачею, яка потребує вирішення у даній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1. Призначення системи

У роботі розглядається побудова платформи з використанням новітніх підходів до розробки розподілених систем прийому, зберігання та обробки великих обсягів даних. Для створення платформи використовувалися сучасні бібліотеки та бази даних з відкритим кодом. Досліджено реалізацію платформи на кластері в хмарному хостингу. Запропоновано архітектуру платформи, яка забезпечує її високу масштабованість і ефективність, а також дозволяє реалізувати рішення для моніторингу мережі пристроїв і роботи з адміністративними даними, які є більш функціональними, ніж існуючі, і мають підвищену безпеку передачі даних.

Метою роботи є розробка серверного рішення для хмарної платформи Інтернету речей з підвищеними вимогами до безпеки передачі даних.

Система повинна забезпечити такі вимоги:

- вирішує основні проблеми, які ставлять подібні платформи (зберігання, аналіз даних тощо), тобто реалізує зазначені підсистеми з урахуванням загальноприйнятих підходів до розробки розподілених систем обробки великих даних [7];

- розроблена з урахуванням передового досвіду та підходів до архітектури програмного забезпечення [8-13], які дозволяють забезпечити функціональні та нефункціональні вимоги до системи;

- має специфічну функціональність, якої немає в інших системах (наприклад, надійна доставка повідомлень між пристроєм і адміністратором і засіб моніторингу пристрою);

- містить інструменти для налаштування хмарного хостингу, а також інструменти для управління групою серверів.

1.2. Область застосування

У даній роботі розглядається побудова серверних рішень, які забезпечують функціонування Інтернету речей.

Зокрема, розглядається побудова наступних підсистем:

- отримання та зберігання даних. Зі швидкістю надходження даних (тобто з мінімальними затримками) він приймає і зберігає їх в умовно постійному сховищі даних (черзі повідомлень);

- Обробка даних. Виконує обробку потокових даних і аналітичні запити. Потокова обробка даних відбувається між умовно постійним сховищем даних і базою даних (постійне сховище даних). Вироджений (no-op) випадок обробки даних, якщо вона не потрібна - простий перепис інформації з черги повідомлень в базу даних;

- аутентифікація та безпека. Вмикає перевірку даних автентифікації відправників (пристроїв) і адміністраторів. Здійснює безпечну передачу інформації між компонентами системи;

- передача адміністративних даних. Він передає дані від пристрою до адміністратора (інформація про стан) і від адміністратора до пристрою (команди). У той же час дані повинні зберігатися в середній підсистемі, оскільки, наприклад, команди повинні зберігатися до тих пір, поки пристрій не запросить їх, оскільки воно може бути тимчасово недоступним для відправки даних безпосередньо на нього;

- моніторинг. Він перевіряє стан пристроїв і інформацію, яку вони надсилають, на основі певних заданих правил.

Отже, виходячи з вищевикладеного, дослідження та програмна реалізація систем керування серверами Інтернету речей з підсистемою безпеки передачі даних є актуальною задачею, яка потребує вирішення в цій магнітній роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

Серверні системи для Інтернету речей (IoT) грають ключову роль у забезпеченні обробки, безпеки, аналізу, збереження та управління даними великої кількості пристроїв та сенсорів, підключених до Інтернету.

Характеристик і властивості серверних систем для IoT:

скейлінг: Системи IoT повинні бути здатними масштабуватися, оскільки кількість підключених пристроїв може бути дуже великою. Серверні рішення повинні бути готовими працювати з великими обсягами даних і обчислювальними завданнями;

обробка в реальному часі: Деякі системи IoT вимагають обробки даних в реальному часі, наприклад, для виявлення аномалій або відправки повідомлень на підставі даних сенсорів;

збереження даних: Інформацію з IoT-пристроїв часто потрібно зберігати для подальшого аналізу або створення звітів. Системи повинні забезпечувати надійне збереження та можливість пошуку даних;

аналіз даних: Важливо мати можливість аналізувати дані, щоб виявляти цінні взорці, використовувати машинне навчання і штучний інтелект для прогнозування подій і прийняття рішень;

моніторинг та управління: серверні системи повинні надавати можливості моніторингу та управління підключеними пристроями IoT, включаючи віддалену настройку та відправку команд;

безпека: Безпека є критичним аспектом для IoT. Системи повинні забезпечувати захист від несанкціонованого доступу до даних та пристроїв;

інтеграція: IoT-серверні системи повинні бути здатні інтегруватися з іншими інформаційними системами в організації, такими як CRM (Customer Relationship Management) або ERP (Enterprise Resource Planning);

підтримка різних протоколів: Враховуючи різноманітність пристроїв IoT, системи повинні підтримувати різні комунікаційні протоколи для взаємодії з різними пристроями;

хмарне обчислення: Багато серверних систем для IoT можуть використовувати хмарні ресурси для обчислення та зберігання даних;

При виборі серверної системи для IoT важливо враховувати потреби конкретного проекту і вибрати рішення, яке відповідає вимогам щодо масштабу, продуктивності, безпеки та інших параметрів.

2.1. Огляд існуючих систем

Інтернет речей (IoT) — це концепція комп'ютерної мережі фізичних об'єктів («речей»), оснащених вбудованими технологіями для взаємодії один з одним або із зовнішнім середовищем, яка розглядає організацію таких мереж як явище, яке може реструктурувати економічні та соціальні процеси і виключають необхідність участі людини в деяких діях та операціях.

Індустрія Інтернету речей стабільно розвивається в останні роки, і її зростання буде тільки прискорюватися. Дослідницько-консалтингова компанія Gartner у своєму звіті [1] передбачила, що у 2025 році у світі буде 6,4 мільярда «речей» — це на 30 відсотків більше, ніж у 2015 році (4,9 мільярда пристроїв). Згідно зі звітом, у грошовому еквіваленті розмір галузі зросте з 1183 мільярдів доларів у 2018 році до 1414 мільярдів доларів у 2023 році. У 2025 році буде вже 20,7 млрд підключених пристроїв, а загальні витрати на них становитимуть 3 трлн доларів (приблизно половина з них - споживачі, інша половина - підприємства).

Найбільший у світі виробник мережевого обладнання Cisco [2] дає ще більш оптимістичний прогноз щодо розміру ринку IoT: 14,4 трильйона доларів до 2026 року. Інтернет речей пронизує всі сфери життя і забезпечує роботу багатьох систем: від суто споживчих (наприклад, різноманітні домашні датчики, носима електроніка, розумні будинки тощо) до промислових (управління та

моніторинг виробничих процесів, розумні енергетичні системи, розумні міста, автономні автомобілі тощо).

За допомогою «речей» додана вартість збільшиться [2] за рахунок:

- покращення клієнтського досвіду;

- скорочення часу, який проходить від проектування товару до його появи в продажу (time-to-market);

- вдосконалення ланцюгів поставок і логістики;

- підвищення продуктивності праці працівників;

- більш ефективне використання коштів (зниження витрат).

Розвиток Інтернету речей сприяє прогресу в багатьох сферах техніки. Зрозуміло, що зростаючий попит на апаратне забезпечення IoT стимулює інновації в електронній промисловості, яка постачає галузь IoT електронні плати, датчики, батареї тощо. Специфіка портативних пристроїв накладає певні обмеження на програмне забезпечення, яке ними керує. Обмежені обчислювальні можливості компакт-дисків, а також вимоги до електроенергії (особливо для «речей»), що живляться від батареї, вимагають економного відношення до ресурсів, а також перенесення майже всіх операцій обробки даних на сторону сервера. Серверні платформи, які обробляють зчитування датчиків, також повинні враховувати, що дані неминуче міститимуть помилки та що пристрої можуть час від часу виходити з ладу та повністю виходити з ладу. Питання безпеки також стає все більш важливим у світі, де велика кількість пристроїв збирає, обробляє та передає різноманітні дані, які мають певну промислову чи комерційну цінність, або містять особисті дані користувачів. Необхідно також реагувати на виклики, пов'язані з постійно зростаючою кількістю інформаційних загроз. Розробка програмних рішень, які керують великою кількістю пристроїв, вимагає використання новітніх підходів у сфері хмарних обчислень, обробки та зберігання великих даних (Big Data). Такі серверні рішення повинні бути горизонтально масштабованими, мати високу пропускну здатність, відмовостійкість і короткий час відповіді на запити.

Наявність різних інструментів аналізу даних має особливе значення для ефективного використання даних в індустрії Інтернету речей. Зараз більшість пристроїв мають виключно інформаційний характер, тобто такі, що лише збирають і передають дані, а не виконують, наприклад, функції керування. Тому інтелектуальна обробка даних є ключовою вимогою для таких систем. Інформацію, зібрану пристроями IoT, наприклад, можна використовувати для оптимізації певних виробничих процесів, контролю якості продукції та моніторингу продуктивності інших систем. Зазвичай він проходить формальний процес обробки, наприклад, агрегацію, а потім переглядається експертами. Подальший розвиток інструментів інтелектуального аналізу даних з використанням методів машинного навчання може зменшити або навіть усунути потребу в участі людини. За наявності ефективних механізмів автоматичного прийняття рішень можна буде збільшити кількість пристроїв, які самостійно виконують функції керування тими чи іншими процесами. Варто також зазначити, що методи та інструменти, які використовуються в серверних рішеннях IoT, є загальними та можуть бути застосовані в інших сферах, де збираються та обробляються великі обсяги даних. Це: аналіз соціальних мереж, аналіз поведінки користувачів веб-сайтів, аналіз стану інфраструктури центрів обробки даних, системи рекомендацій, націлювання реклами, веб-пошук тощо.

Платформи IoT

Зараз у цій сфері працює кілька платформ. Найвідоміші з них: Amazon Web Services IoT Platform, Google Cloud IoT Platform, ThingWork, Oracle IoT, IBM IoT Platform. Здебільшого існуючі рішення пропонують загальний набір функціональних можливостей, який включає зберігання даних, а також досить широкі можливості для їх відображення та аналізу. Усі зазначені платформи є комерційними та закритими, що є їх суттєвим недоліком. Така закритість ускладнює їх розширення, адаптацію та оптимізацію для конкретного випадку використання. Компанії, які використовують платформи, що працюють за цією моделлю, часто прив’язані до постачальника (прив’язка до постачальника) і не

можуть переорієнтуватися на використання іншої платформи без значних витрат часу та грошей. Платформ з відкритим кодом майже немає. Однак цілком можливо розробити функціональну платформу, яка використовуватиме компоненти та бібліотеки з відкритим кодом, оскільки наразі існує багато рішень із відкритими ліцензіями, які можуть стати частиною системи та є вже стабільними та багатофункціональними.

У цьому документі були досліджені, порівняні та використані відповідні компоненти. Існуючі платформи також лише іноді мають готові рішення для очищення помилкових даних. Однак загалом такі платформи можуть запропонувати базові інструменти статистики або машинного навчання, які підходять майже для всіх випадків використання. Однак варто зазначити, що очищення даних є великою та складною частиною науки про дані. Виявлення аномалій є особливо перспективним напрямком. Онлайн-виявлення аномалій наразі є однією з найменш досліджених областей машинного навчання.

У роботі також розглядається організація моніторингу та передачі адміністративних даних: команд від адміністратора до пристрою та інформації про стан від пристрою (або сервера моніторингу) до адміністратора. Цей функціонал дуже важливий для ефективного управління великими групами пристроїв, тоді як існуючі платформи майже не реалізують такі можливості.

На відміну від існуючих платформ, розроблене рішення спрямоване на універсальність: замість використання конкретних протоколів для передачі даних і надання SDK для пристроїв кожного окремого виробника, воно використовує загальноприйняті веб-протоколи, для яких існує багато бібліотек, що прискорює та знижує витрати на клієнт. розробка програмного забезпечення (тобто, яке працює на пристроях і передає дані в мережу). Також універсальність розробленого рішення полягає в забезпеченні можливості спеціалізації платформи під конкретний сценарій. Наприклад, дані повинні зберігатися для всіх типів пристроїв. Але для різних сфер використання пристрою обробка цих даних буде різною, тому доцільно передбачити можливість розширення

платформи, вказавши певні ланцюжки операцій обробки даних для конкретних потреб.

Amazon Web Services (AWS IoT)

AWS IoT це частина Amazon Web Services, яка надає послуги для підключення пристроїв IoT, збору, аналізу та обробки даних, а також безпеки.

Вона підтримує різні пристрої та протоколи і має інструменти для відправки повідомлень, моніторингу та аналізу даних.

Складається із таких сервісів

AWS IoT Core є центральною послугою для підключення IoT-пристроїв. Вона підтримує різні протоколи, включаючи MQTT, HTTP, та WebSocket.

Пристрої можуть бути зареєстровані в AWS IoT Core, і вони можуть взаємодіяти з центральним сервером AWS IoT для відправки даних та приймання команд.

AWS IoT Device Management

Ця служба надає можливості для керування та віддаленого налаштування IoT-пристроїв.

Ви можете відслідковувати стан і статистику кожного пристрою, встановлювати правила і виконувати дії в залежності від змін стану пристроїв.

AWS IoT Analytics

AWS IoT Analytics дозволяє вам аналізувати дані, які збираються з IoT-пристроїв.

Ви можете створювати різні пайплайни для обробки даних, аналізувати їх і створювати звіти та візуалізації.

AWS IoT Events

AWS IoT Events служить для виявлення аномалій і створення реакцій на події.

Ви можете налаштовувати правила для спостереження за даними та генерації сповіщень та дій в реальному часі.

AWS IoT Greengrass

AWS IoT Greengrass дозволяє виконувати обчислення та обробку даних на краю (edge) близько до IoT-пристроїв, зменшуючи завантаження на хмарні ресурси.

Ви можете створювати локальні застосунки і виконувати їх на пристроях Greengrass.

AWS IoT Device Defender

Ця послуга пропонує засоби для забезпечення безпеки IoT-пристроїв. Вона дозволяє виявляти та запобігати загрозам безпеці, а також перевіряти настройки безпеки пристроїв.

AWS IoT SiteWise

AWS IoT SiteWise спрямований на збір, обробку та аналіз даних з промислових пристроїв і обладнання.

Ви можете створювати візуалізації та аналізувати дані, щоб оптимізувати виробництво та управління активами.

Хмарне зберігання та інтеграція

AWS IoT легко інтегрується з іншими послугами AWS, такими як Amazon S3, Amazon Kinesis, Amazon Lambda та іншими, для зберігання даних, обчислення та інтеграції.

Безпека

AWS IoT надає рішення для захисту даних, аутентифікації пристроїв, керування доступом і шифрування.

AWS IoT - це потужна і розширювана платформа, яка надає повний спектр інструментів для розробки та управління проектами IoT в різних галузях. Вона дозволяє спростити розробку та впровадження проектів IoT, забезпечуючи високий рівень безпеки та можливості аналізу даних.

Microsoft Azure IoT

Azure IoT - це рішення від Microsoft для створення і впровадження проектів IoT.

Вона включає в себе IoT Hub для підключення пристроїв, Stream Analytics для аналізу даних, та цілий ряд інших служб.

Azure IoT Hub

Azure IoT Hub є центральною послугою Azure для підключення, керування та моніторингу IoT-пристроїв. Вона підтримує різні протоколи, включаючи MQTT, HTTP і AMQP.

За допомогою IoT Hub ви можете реєструвати, керувати та моніторити ваші IoT-пристрої, а також передавати дані в Azure для подальшої обробки.

Azure IoT Central

Azure IoT Central - це високорівнева платформа для швидкого створення рішень IoT без значних зусиль і навичок в розробці.

Вона надає засоби для створення готових до використання застосунків IoT з вбудованими можливостями моніторингу та аналізу даних.

Azure Stream Analytics

Azure Stream Analytics дозволяє аналізувати та обробляти потокові дані, які надходять від IoT-пристроїв.

Ви можете налаштовувати правила, фільтри, агрегацію даних і відправку оброблених даних до інших служб Azure.

Azure IoT Edge

Azure IoT Edge надає можливість виконувати обчислення та обробку даних на краю (edge) близько до IoT-пристроїв, зменшуючи завантаження на хмарні ресурси.

Ви можете створювати локальні застосунки і виконувати їх на пристроях IoT Edge.

Azure Time Series Insights

Azure Time Series Insights - це інструмент для аналізу часових рядів даних з IoT-пристроїв.

Він надає можливості візуалізації, аналізу та створення звітів на основі даних часових рядів.

Azure IoT Solutions

Microsoft пропонує готові рішення IoT для різних галузей, такі як медицина, виробництво, транспорт і багато інших.

Ці рішення включають готові до використання сценарії, візуалізації та інші компоненти для розробки проектів IoT.

Azure IoT Security

Microsoft присвятив особливу увагу безпеці в сфері IoT. Azure IoT пропонує заходи безпеки, включаючи аутентифікацію, шифрування та керування доступом до пристроїв.

Інтеграція з іншими службами Azure

Azure IoT легко інтегрується з іншими послугами Azure, такими як Azure Machine Learning, Azure Functions, Azure Logic Apps та багато інших.

Підтримка різних пристроїв і платформ

Azure IoT підтримує широкий спектр пристроїв і платформ, включаючи мікроконтролери, Linux і Windows, а також підключення до різних облікових записів в інших публічних хмарних сервісах.

Глобальні можливості масштабування

Azure IoT може легко масштабуватися від декількох пристроїв до мільйонів, забезпечуючи надійну та високодоступну інфраструктуру для вашого проекту IoT.

Azure IoT - це потужна та гнучка платформа для розробки і впровадження проектів IoT з багатьма можливостями та інструментами для забезпечення успішності вашого проекту.

Google Cloud IoT

Google Cloud пропонує послуги для підключення, обробки і аналізу даних IoT. Це включає в себе Cloud IoT Core для керування пристроями та Cloud Pub/Sub для передачі даних.

Google Cloud IoT - це набір послуг та інструментів, розроблений Google для розробки та впровадження проектів Інтернету речей (IoT) на платформі Google Cloud. Ось детальний огляд ключових компонентів Google Cloud IoT:

Cloud IoT Core

Cloud IoT Core є центральною послугою Google для підключення та керування IoT-пристроями. Вона підтримує протоколи MQTT та HTTP.

Ви можете легко підключати свої IoT-пристрої до Cloud IoT Core та керувати ними, встановлювати правила для передачі даних в інші послуги Google Cloud.

Cloud IoT Device Manager

Cloud IoT Device Manager надає можливості для керування IoT-пристроями та їх конфігурацією.

Ви можете створювати, керувати і відслідковувати статус всіх підключених пристроїв.

Cloud Pub/Sub

Cloud Pub/Sub дозволяє передавати дані з IoT-пристроїв в різні служби Google Cloud для подальшої обробки та аналізу.

Ви можете налаштовувати правила маршрутизації для обробки даних та створення звітів.

Cloud Dataflow

Cloud Dataflow - це послуга для обробки потокових даних, включаючи дані з IoT-пристроїв.

Вона дозволяє аналізувати, обробляти та агрегувати дані в режимі реального часу.

Cloud Bigtable

Cloud Bigtable - це розподілена база даних, призначена для збереження та аналізу великих обсягів даної IoT-інформації.

Вона дозволяє ефективно зберігати дані та виконувати запити до них.

Cloud Machine Learning Engine

Cloud Machine Learning Engine надає інструменти для розробки та впровадження моделей машинного навчання для аналізу даних IoT.

Ви можете створювати та навчати моделі для виявлення взорців та прогнозування подій.

Інтеграція з іншими службами Google Cloud

Google Cloud IoT легко інтегрується з іншими послугами Google Cloud, такими як Google Cloud Storage, Google Cloud Functions, BigQuery і багато інших.

Безпека

Google надає рішення для забезпечення безпеки IoT-пристроїв та даних, включаючи аутентифікацію, шифрування та контроль доступу.

Глобальні можливості масштабування

Google Cloud IoT забезпечує глобальну масштабовану інфраструктуру, що дозволяє вам легко масштабувати свій проект від мільйонів IoT-пристроїв.

Google Cloud IoT - це потужна та розширювана платформа для розробки і впровадження проектів IoT з багатьма можливостями та інструментами для забезпечення успішності вашого проекту.

IBM Watson IoT

IBM Watson IoT надає інструменти для розробки та управління застосунками IoT.

Вона включає в себе Watson IoT Platform для керування підключеними пристроями та Watson Studio для аналізу даних.

IBM Watson IoT - це набір інструментів та послуг для розробки, впровадження та управління проектами Інтернету речей (IoT) від IBM. Ця платформа надає рішення для збору, аналізу, візуалізації та управління даними, отриманими від підключених пристроїв. Ось детальний огляд ключових компонентів IBM Watson IoT:

Watson IoT Platform

Watson IoT Platform є центральною частиною платформи і надає можливості для підключення, реєстрації та керування IoT-пристроями.

Ви можете відслідковувати стан кожного пристрою, відправляти команди та встановлювати правила для обробки даних.

Watson IoT Platform Analytics

Ця послуга надає можливості для аналізу та обробки даних з IoT-пристроїв, включаючи машинне навчання та аналіз часових рядів.

Ви можете створювати прогнози, виявляти аномалії та оптимізувати роботу вашого обладнання.

Watson IoT Platform Operations Console

Ця послуга надає інструменти для керування вашими IoT-пристроями та даними через веб-інтерфейс.

Ви можете створювати сповіщення, візуалізації та встановлювати правила.

Watson IoT Platform Blockchain

IBM Watson IoT може інтегруватися з технологією блокчейн для забезпечення безпеки та надійності даних з IoT-пристроїв.

Це особливо корисно в галузях, де важлива доказова база даних.

Watson IoT Platform Edge Analytics:

Ця послуга дозволяє вам виконувати обробку даних на краю (edge) близько до IoT-пристроїв.

Ви можете створювати локальні застосунки для аналізу даних без необхідності передачі їх в хмару.

Watson IoT Platform Device Lifecycle Management

Ця послуга надає інструменти для ефективного керування життєвим циклом IoT-пристроїв, включаючи реєстрацію, конфігурацію та управління оновленнями програмного забезпечення.

Watson IoT Continuous Engineering

IBM також пропонує інструменти для інженерії продукту, які можуть бути корисні в розробці пристроїв IoT.

Інтеграція з іншими послугами IBM

Watson IoT може легко інтегруватися з іншими продуктами та послугами IBM, такими як Watson AI, IBM Cloud, IBM Maximo, і інші.

Cisco IoT Cloud Connect

Cisco пропонує рішення для підключення та управління пристроями IoT.

Це включає в себе мережеве обладнання та платформи для забезпечення безпеки та аналізу даних IoT.

ThingSpeak

ThingSpeak - це безкоштовна платформа для IoT від MathWorks (розробника MATLAB).

Вона дозволяє відправляти дані з пристроїв IoT, візуалізувати дані та створювати сповіщення.

Збір даних IoT

ThingSpeak дозволяє підключати інтернет-пристрої до платформи та надсилати дані на неї через різні протоколи, такі як MQTT, HTTP, або ThingHTTP.

Ви можете легко інтегрувати свої IoT-пристрої та мікроконтролери, щоб відправляти дані на ThingSpeak.

Збереження та візуалізація даних

ThingSpeak зберігає надходячі дані та надає можливості для їх візуалізації у вигляді графіків, графіків та таблиць.

Ви можете налаштовувати спеціальні панелі і візуалізації, щоб відображати дані у зручному для вас форматі.

Аналіз даних

ThingSpeak має інтегровані інструменти для аналізу та обробки даних. Ви можете використовувати MATLAB для створення власних аналітичних скриптів.

Це дозволяє вам виявляти взорці, прогнозувати значення та вживати інші аналітичні методи.

Сповіщення та інтеграція

ThingSpeak підтримує сповіщення на основі умов. Ви можете налаштовувати сповіщення, коли визначені умови виконуються.

Також, ви можете інтегрувати ThingSpeak з іншими веб-сервісами та платформами, відправляти дані до зовнішніх додатків або обробляти їх на інших сервісах.

Відкритий доступ до даних

ThingSpeak дозволяє вам встановлювати права доступу до вашої інформації. Ви можете визначати, хто має право читати та записувати дані.

Безкоштовний план та публічні канали:

ThingSpeak пропонує безкоштовний план, який включає обмеження, але може бути корисним для початкового використання та вивчення IoT.

Також, ви можете розміщувати дані на публічних каналах ThingSpeak, які доступні для загального перегляду та використання.

Легка інтеграція з MATLAB і Simulink

ThingSpeak інтегрується з іншими інструментами MathWorks, такими як MATLAB і Simulink, що робить його ідеальним вибором для тих, хто вже працює з цими інструментами.

ThingSpeak - це зручний інструмент для розробки простих проектів IoT, збору даних і візуалізації інформації. Він також підходить для навчання та демонстраційних цілей.

PlatformIO

PlatformIO - це відкрите середовище розробки для IoT, яке підтримує багато мікроконтролерів і платформ.

Вона надає інструменти для написання, тестування та відлагодження програм для IoT-пристроїв.

PlatformIO - це інтегроване середовище розробки (IDE) та платформа для роботи з вбудованими системами та мікроконтролерами. Це потужний інструмент для розробників, які працюють з платформами Інтернету речей (IoT),

мікроконтролерами та одноплатними комп'ютерами. Ось детальний огляд ключових характеристик та можливостей PlatformIO:

Підтримка різних мікроконтролерів та платформ

PlatformIO підтримує широкий спектр мікроконтролерів та платформ, включаючи Arduino, ESP8266, ESP32, Raspberry Pi, STM32, AVR та багато інших.

Ви можете працювати з різними мікроконтролерами у одному проекті.

Компіляція та завантаження коду:

PlatformIO надає інструменти для компіляції, завантаження та відлагодження коду безпосередньо на мікроконтролери.

Ви можете легко оновлювати прошивку пристроїв через USB або в мережі.

Підтримка мов програмування

PlatformIO підтримує різні мови програмування, включаючи C, C++, Python, Rust та інші.

Ви можете вибрати мову, яка найкраще відповідає вашим потребам та навичкам.

Бібліотеки та ресурси

Платформа надає доступ до багатьох бібліотек та ресурсів для розробки вбудованих систем. Ви можете легко встановлювати бібліотеки та ресурси через менеджери пакетів.

Відлагодження коду

PlatformIO дозволяє відлагоджувати код за допомогою відлагоджувача GDB.

Ви можете встановлювати точки зупинки, аналізувати змінні та відлагоджувати код в режимі реального часу.

Інтегровані засоби роботи з версіями

PlatformIO інтегровано з системами контролю версій, такими як Git. Ви можете вести історію змін та спільно працювати над проектами з іншими розробниками.

Підтримка платформи IoT та хмарних сервісів

PlatformIO підтримує розробку для платформ Інтернету речей (IoT) та інтеграцію з хмарними сервісами, такими як Amazon Web Services, Google Cloud і інші.

Кросплатформеність та відкритий джерело

PlatformIO доступний для різних операційних систем, включаючи Windows, macOS та Linux.

Він є відкритим проектом, тому розробники можуть призначати внесок до його розвитку.

PlatformIO - це важливий інструмент для розробників вбудованих систем та IoT-проектів, який спрощує розробку, тестування та впровадження коду на різних мікроконтролерах та платформах.

Кожна з цих платформ має свої особливості та переваги. Вибір платформи залежить від конкретних потреб вашого проекту, ваших технічних знань та ресурсів, які ви готові вкласти.

Бажано провести ретельний аналіз і тестування перед тим, як вибирати оптимальну платформу для вашого проекту IoT.

2.2 Обґрунтування вибору методів розробки

Для дослідження та реалізації програмного забезпечення побудови платформи Інтернету речей були обрані наступні інструменти та мови програмування:

– запропоновано та реалізовано таку розподілену сервіс-орієнтовану (мікросервісну) архітектуру платформи, яка забезпечує високу продуктивність і

майже нескінченну масштабованість. Це дозволяє збільшити потужність системи для обслуговування мереж пристроїв будь-якого розміру та збільшити можливості підсистеми для аналізу даних. Ефективне впровадження компонентів системи дозволяє оптимізувати витрати на обладнання платформи;

- під час розробки платформи були обрані та обґрунтовано застосовані сучасні системи, протоколи, формати даних, бібліотеки та бази даних. Рішення незалежних виробників мають відкритий код, що спрощує розширення системи, має економічні та інші переваги;

- детально описано різні аспекти обраних архітектурних рішень: від обґрунтування високорівневого поділу на функціональні модулі до характеристик ефективної реалізації введення/виведення та впливу обраної схеми автентифікації на масштабованість платформи. архітектура;

– на відміну від існуючих платформ, розроблене рішення є добре розширюваним у тому сенсі, що його можна легко адаптувати для аналізу даних, що надходять від різних пристроїв і датчиків, шляхом визначення ланцюжків обробки даних. Протоколи та формати даних були обрані не лише через їхню ефективність, а й через зручність і простоту розширення платформи.

Для вирішення поставлених завдань використовувалося наступне обладнання та мови програмування:

Практична реалізація платформи виконана на Java (версія 8.1) та Scala.

Використовувалися такі бібліотеки:

- Undertow — це веб-сервер, написаний мовою Java, який дуже ефективний завдяки використанню неблокувального вводу-виводу. Він дозволяє описати логіку за допомогою окремих обробників, які обслуговують запити на певну адресу. За допомогою цієї бібліотеки реалізовані сервери переднього плану, тобто інтерфейси HTTP (REST API) і WebSocket;

- Kafka clients – клієнтські інтерфейси для взаємодії з кластером Apache Kafka;

- набір бібліотек та інтерфейсів для роботи з Apache Spark;

- Конектор Spark-Cassandra – драйвер Apache Cassandra з інтерфейсом, інтегрованим з API Apache Spark;

- MongoDB Async Driver – драйвер MongoDB, реалізований за допомогою Java NIO для неблокуючих запитів до бази даних;

- Бібліотека для створення та перевірки веб-токенів JSON;

- Apache Commons Codec – набір реалізацій кодеків, наприклад, Base64 для формування веб-токенів JSON;

- SLF4J – Java API для запису різноманітних інформаційних та діагностичних повідомлень про хід програми;

- Logback – реалізація SLF4J;

- JUnit – бібліотека модульного тестування.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на магістерську роботу, реалізації підлягає програмне забезпечення, яке призначено для серверної системи управління IoT.

В процесі розробки магістерської роботи 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

Розглянемо основні компоненти системи, потоки даних між ними, а також її входи та виходи.

На рисунку 3.1 також показано деякі сторонні системи та бібліотеки (наприклад, бази даних), які будуть використовуватися в певному компоненті. Їх вибір буде обґрунтовано в наступних розділах.

На вході системи (блок «Джерела даних») ми маємо різні пристрої, такі як датчик температури, датчик тиску, датчик якості повітря або будь-які інші. Вони надсилають дані на набір передніх серверів (блок «Вхідні сервери»).

Разом із даними пристрої передають інформацію автентифікації та інформацію про себе (наприклад, ідентифікатор пристрою, за яким можна визначити тип пристрою).

Внутрішні сервери виконують перевірку інформації автентифікації. Якщо воно неправильне (тобто відправник невідомий), дані відхиляються. Відповідно до інформації пристрою відбувається завантаження даних на зовнішні сервери: визначається тип повідомлення та надсилається у відповідну тему в черзі повідомлень (блок «Apache Kafka»).

Внутрішні сервери також перевіряють цілісність даних (валідують). (блок «Apache Kafka»).

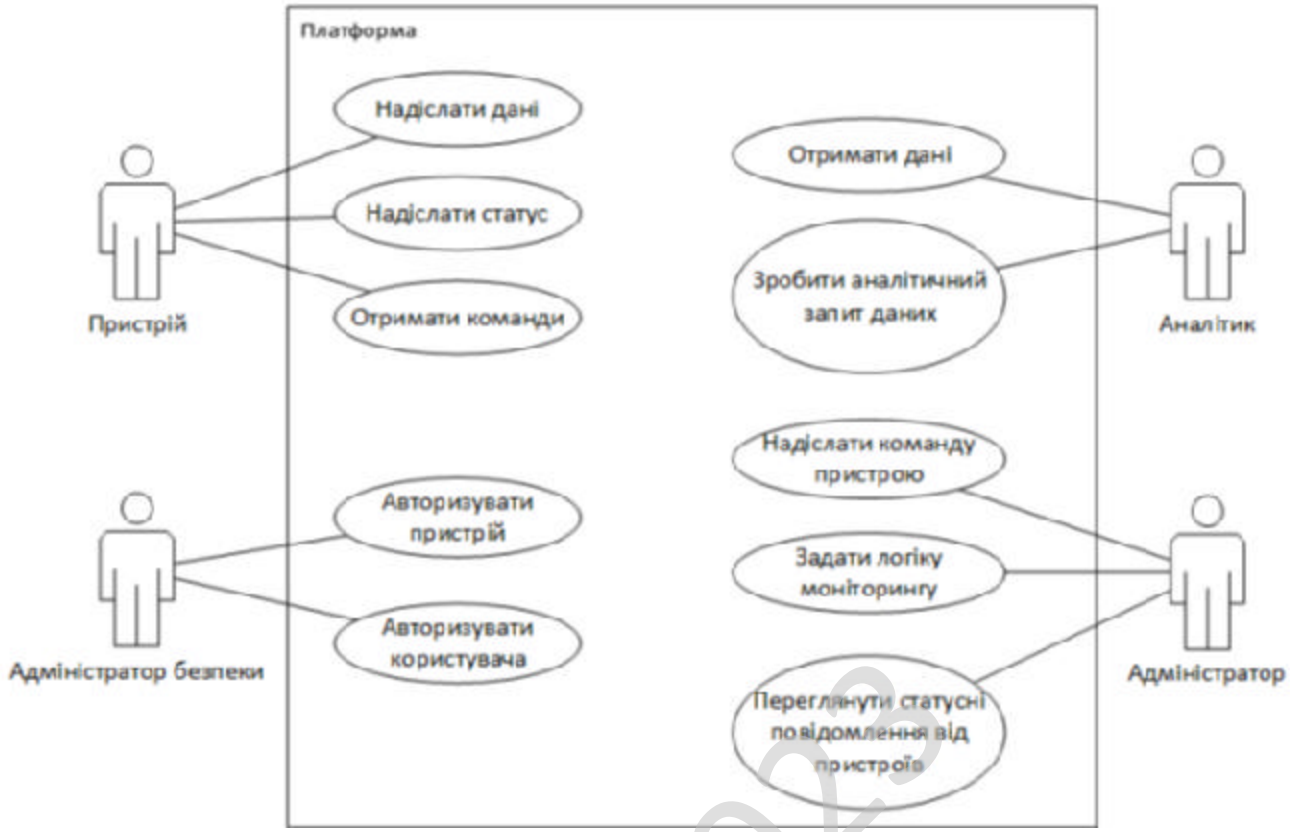

Рисунок 3.1 - Взаємодію із загальною функціональністю платформи

3.2 Розробка структурної схеми

Структурна схема системи – це сукупність об’єктів та частин та взаємозв’язки між ними.

Призначенням структурної схеми є наглядне відображення складових частин розробляємої системи, її основних блоків, вузлів та взаємозв’язок між ними.

Виділимо чотири основні групи користувачів системи. У нас є пристрій (або датчик), який надсилає дані в систему, яка обробляється та зберігається; надсилає повідомлення про статус, які потім отримують адміністратори; приймає команди, надіслані адміністраторами.

Адміністратори також встановлюють логіку моніторингу. Адміністратор безпеки вводить дані для автентифікації пристрою та інших користувачів системи.

Адміністраторам і аналітикам призначаються права доступу, які показують, які саме пристрої вони можуть контролювати або отримувати дані. Аналітики можуть запросити готові дані, які вже зберігаються та попередньо оброблені в системі, або сформувати аналітичний запит, який вимагає виконання певних обчислень на існуючих даних.

Структурна схема системи IoT зображена на рисунку 3.2 .

Рисунок 3.2 – Структурна схема систем IoT

3.3 Розробка функціональної схеми

Як видно з блок-схеми, існує кілька пристроїв на різних платформах, які надсилають дані на сервер певним чином. Деякі пристрої мають вбудовані карти, які дозволяють надсилати дані безпосередньо на сервер.

Наприклад, пристрій може мати дротове підключення до Інтернету через Ethernet або доступ до бездротової точки доступу (Wi-Fi), яка забезпечує доступ до Інтернету. Часто, якщо кілька пристроїв розташовані на певній території або в будівлі на невеликій відстані, вони безпосередньо підключаються до шлюзу (кількість шлюзів значно менше кількості датчиків), який, у свою чергу, забезпечує передачу даних на сервер через Інтернет.

Це робиться з різних причин:

- економічний: наявність складного мережевого обладнання збільшує вартість пристрою;

- енергоспоживання: спеціалізовані протоколи передачі даних, такі як Bluetooth Low Energy, забезпечують більш ефективне використання енергії, що особливо важливо, якщо пристрої працюють від автономного живлення;

- програмне забезпечення: пристрої можуть передавати дані на шлюз у довільному двійковому форматі, який легко розробити та мінімізує час процесора. Шлюз, у свою чергу, перетворює дані в певний загальноприйнятий (наприклад, JSON або Protobuf) формат і передає їх на сервер. Крім того, мережу від пристрою до шлюзу можна вважати безпечною та передавати відкриті дані, а на шлюзі можна шифрувати та надсилати в мережу за допомогою безпечного протоколу (наприклад, SSL).

Отже, після того, як дані потрапляють на сервер, вони проходять певну попередню обробку та зберігаються. Потім ці дані доступні для подальшого аналізу.

У цій роботі розглядається побудова серверної платформи. Конкретні питання розробки програмного забезпечення для пристроїв і шлюзів (прошивки) значною мірою виходять за рамки цього проекту: описуються лише основні

питання реалізації клієнтської частини платформи. Також важливо сформулювати нефункціональні вимоги до платформи:

- адаптивність. Збільшення обчислювальних ресурсів має відповідно збільшити пропускну здатність системи (залежність має бути максимально наближеною до лінійної). Крім того, щоб відповідати вимогам систем великих даних і хмарних додатків, масштабованість повинна бути горизонтальною, тобто такою, щоб можна було збільшити продуктивність системи шляхом збільшення кількості вузлів у хмарному центрі обробки даних.

Горизонтальне масштабування також набагато економічніше, ніж вертикальне масштабування.

- відмовостійкість. Необхідно забезпечити таку архітектуру системи, яка має запасні компоненти (реплікацію), а також здатна реагувати на збої та перевантаження під час роботи без повної зупинки служби. Відмовостійкість особливо важлива для тих частин системи, які забезпечують прийом і зберігання даних. Під час недоступності сервера пристрої просто не можуть зберігати свої дані протягом тривалого часу, а потім передавати їх на сервер через обмежений обсяг пам'яті.

- **Продуктивність**. Впровадження системи має бути максимально ефективним, щоб забезпечити пропускну здатність на рівні сотень тисяч запитів на секунду і більше для підсистеми прийому та обробки інформації.

Аналітичні запити також потрібно обробляти в досить короткий час (не більше кількох хвилин).

Функціональна схема розробленої системи зображена на рисунку 3.3.

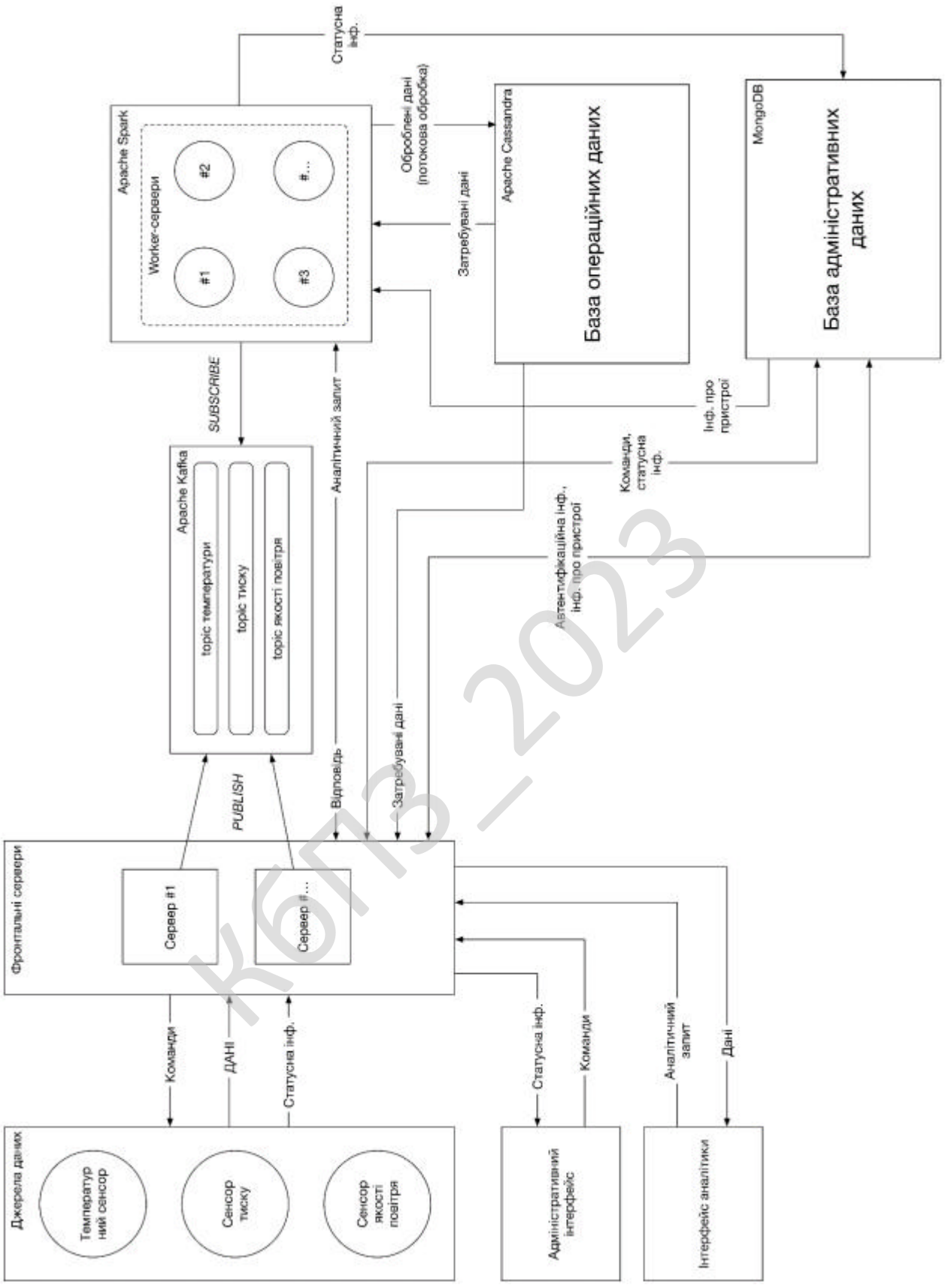

Рисунок 3.3 - Функціональна схема системи

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.4. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

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.

Використовуючи існуючи методи була створена система управління розумним будинком з підсистемою забезпечення безпеки передачі управляючих сигналів від смартфона до мікроконтролера, яка забезпечує безпеку передачі даних при управління системою віддалено з використанням технології Internet.

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.

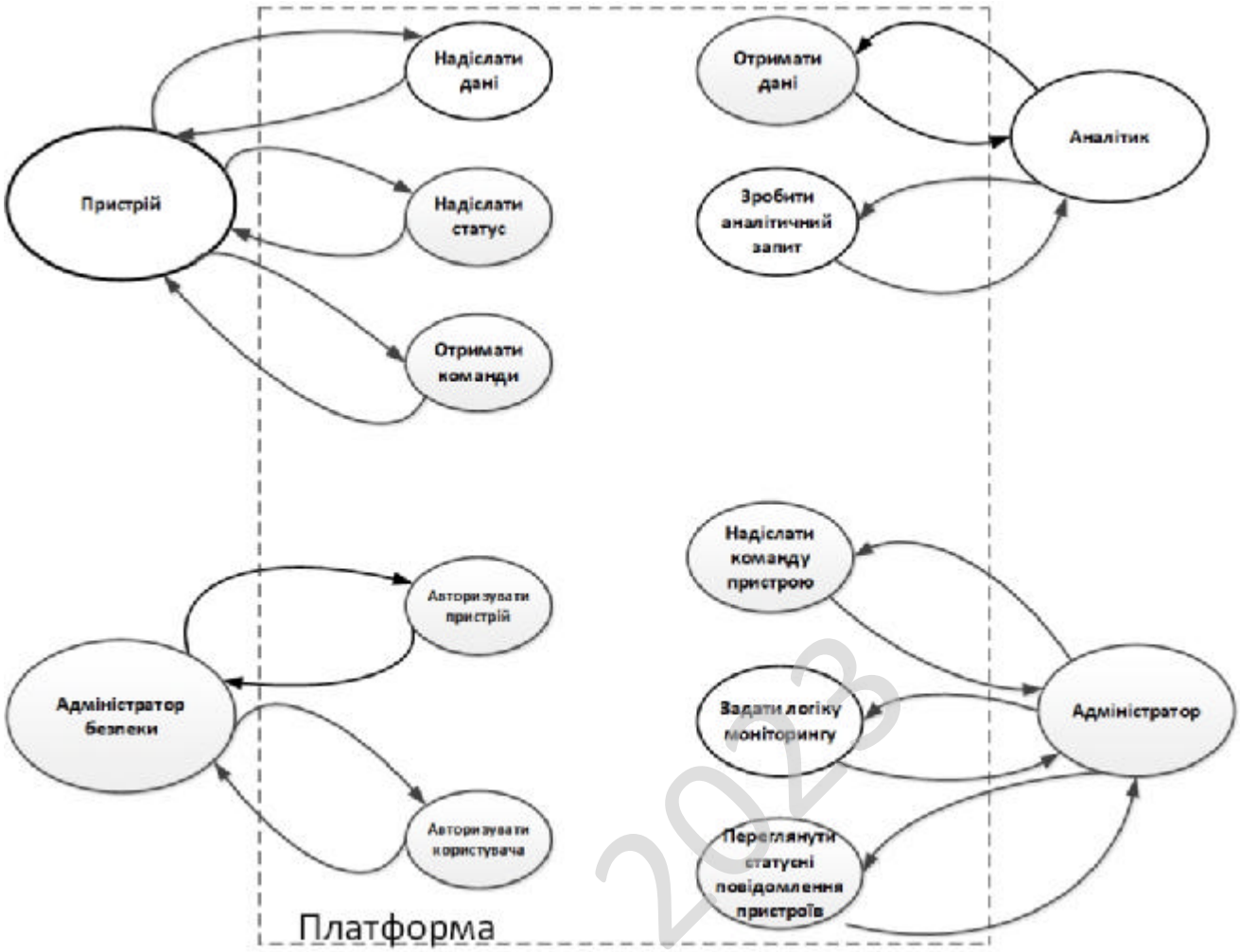

Рисунок 3.3 – Діаграма процесів системи

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ

Опишемо основні компоненти системи, потоки даних між ними, а також її входи та виходи.

На вході системи (блок «Джерела даних») ми маємо різні пристрої, такі як датчик температури, датчик тиску, датчик якості повітря або будь-які інші. Вони надсилають дані на набір передніх серверів (блок «Вхідні сервери»). Разом із даними пристрої передають інформацію автентифікації та інформацію про себе (наприклад, ідентифікатор пристрою, за яким можна визначити тип пристрою).

Внутрішні сервери виконують перевірку інформації автентифікації. Якщо воно неправильне (тобто відправник невідомий), дані відхиляються. Відповідно до інформації пристрою відбувається завантаження даних на зовнішні сервери: визначається тип повідомлення та надсилається у відповідну тему в черзі повідомлень (блок «Apache Kafka»). Внутрішні сервери також перевіряють цілісність даних (валідують).

Черга повідомлень реалізує шаблон дизайну публікації-підписки. Інтерфейсні сервери публікують дані в Apache Kafka. На стороні підписки є система обробки даних – Apache Spark. Здійснює первинну обробку даних, які згодом зберігаються в оперативній базі даних. Обробка даних на цьому етапі може бути чисто потоковою або мікропакетною (мікропакетною). Мікропакетна обробка може емулювати потокову обробку, таким чином дозволяючи застосовувати інструменти та логіку пакетної обробки. Також мікропакетна обробка природно відповідає логіці агрегації та віконної обробки, коли, наприклад, необхідно зібрати певну кількість показань за хвилину, знайти їх середнє значення та зберегти в базі даних. Ви також можете застосувати фільтрацію (або виявлення аномалії) на рівні мікропакета.

База даних, яка містить оперативні дані, забезпечує постійне зберігання оброблених даних. З точки зору продуктивності, це сховище має бути оптимізоване для запису, оскільки операцій читання, очевидно, значно менше. Однак черга повідомлень служить буфером у випадку, якщо операційна база даних не в змозі (наприклад, під час стрибка навантаження) зберігати дані зі швидкістю надходження. Слід зазначити, що ця база даних не вимагає таких значних гарантій, як ті, що надаються традиційними транзакційними базами даних. Наразі буде описана послідовність перенесення інформації з пристрою на постійне сховище. Візуально потоки даних у цьому випадку використання показані на схемі (рисунку 4.1).

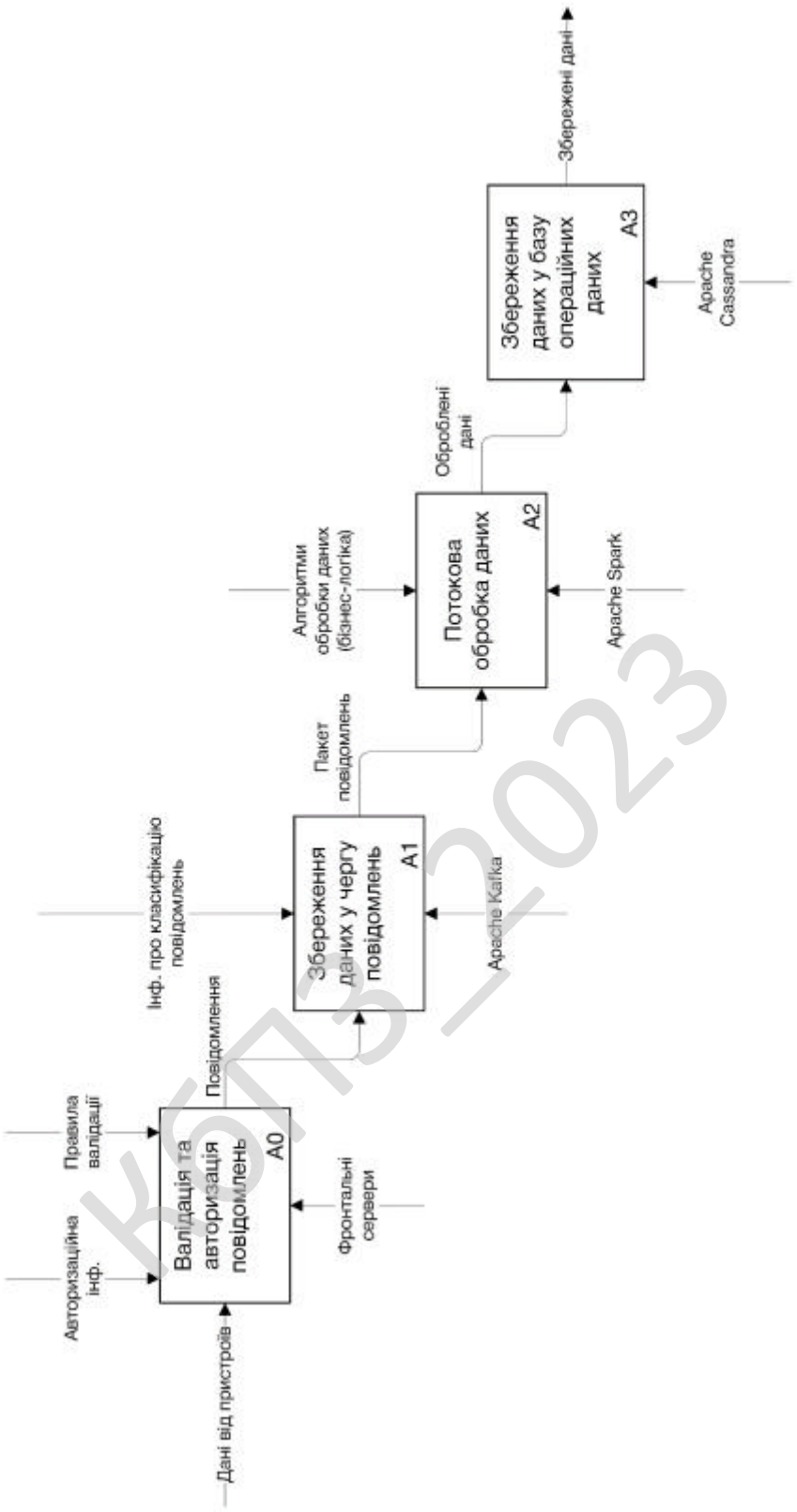

Рисунок 4.1 – Діаграма потоків і процесів при отриманні даних від сенсорів

Опишемо передачу інформації від пристрою до адміністратора. Такою інформацією може бути певна інформація про стан, наприклад, стан акумуляторів пристрою або певні сповіщення про проблеми.

Інформація про статус надсилається з пристрою на фронт-енд-сервери, які, як і при отриманні оперативних даних, підтверджують і аутентифікують одержувача запиту.

Після цього дані надсилаються до адміністративної бази даних. У цьому випадку зазвичай потрібні значні гарантії постійного зберігання та узгодженості даних. Таким чином, після того, як зовнішній сервер надсилає дані до БД, він очікує підтвердження отримання та збереження, перш ніж надсилати відповідне підтвердження на пристрій, який може реалізувати певну логіку повторної спроби, якщо повідомлення є важливим. Після збереження цієї інформації адміністратор надсилає запит на зовнішні сервери, які перевіряють його права доступу до цих даних і роблять запит до бази даних відповідно до предикату запиту (наприклад, повідомлення про стан пристрою із зазначеним ідентифікатором).

Іншим джерелом інформації про стан є механізми моніторингу. Адміністратори у вигляді правил встановлюють певні умови, які повинні бути дотримані під час очікуваної роботи пристрою, і визначають порядок їх перевірки. Наприклад, адміністратор може визначити правило, згідно з яким певний пристрій має надсилати принаймні певну кількість повідомлень за одиницю часу. Якщо ця умова не виконується, тобто активується певний тригер, платформа надсилає повідомлення про статус, щоб повідомити адміністратора про можливу проблему. Подібні умови можна перевірити в різних частинах системи. Внутрішні сервери можуть відстежувати стан з’єднання з пристроями. Підсистема обробки даних може періодично (за розкладом) опитувати сховище оперативних даних для виявлення потенційно некоректних даних, спричинених збоєм пристрою. Також опишемо передачу команд від адміністратора на пристрій. Команда адміністратора надсилається на зовнішній сервер, який

надійно зберігає її в адміністративній базі даних. Пристрої періодично опитують зовнішні сервери щодо нових команд. Потоки даних показано на рисунку 4.2. Така архітектура необхідна для надійної доставки даних на пристрої, оскільки вони можуть бути тимчасово недоступні (вимкнені або не мають зв'язку з сервером). Загалом можна було б зменшити затримку передачі даних і заощадити ресурси, надсилаючи дані безпосередньо (якщо це можливо). Наприклад, якщо зовнішній сервер має відкрите підключення до пристрою (наприклад, через WebSocket), то при отриманні команди від адміністратора її можна відправити негайно. Але це зменшить гарантії доставки. Якщо команда записана в базі даних, пристрій може отримати її, виконати і лише після цього відправити підтвердження на фронтенд-сервер, який позначить команду як прийняту або видалить її. Якщо команду надіслано безпосередньо на пристрій, він може підтвердити отримання, але якщо під час виконання станеться помилка, пристрій не зможе знову запитати команду з зовнішнього сервера, оскільки вона не буде збережена. Якщо ви не підтвердите отримання команди перед її виконанням, то адміністратору доведеться довго чекати завершення свого запиту, що погіршить його сприйняття інтерфейсу в порівнянні з асинхронною схемою, коли він отримує підтвердження відразу після того, як його команда буде збережена в базі даних. Тепер опишемо дії, які відбуваються при отриманні аналітичних запитів. Інтерфейсні сервери автентифікують запит і визначають його тип. Запит може бути задоволений доступними даними або може вимагати додаткових обчислень. Існуючі дані - це ті, що пройшли первинну обробку та зберігаються в оперативній базі даних. Для цього типу запиту достатньо прочитати відповідні дані та надіслати їх. Якщо надійшов аналітичний запит, то необхідно сформувати план його виконання, зчитати дані з однієї або обох баз даних, а потім обробити їх. Якщо потрібна лише додаткова обробка наявних даних, то на вхід системи для обробки подається лише інформація з оперативної бази даних. Адміністративну базу даних можна використовувати, якщо запит вимагає читання певної інформації про пристрій. Наприклад, аналітику можуть

знадобитися зведені дані від датчиків у певній області. У цьому випадку необхідно спочатку здійснити пошук в адміністративній базі ідентифікаторів пристроїв на досліджуваній території, а потім за отриманими ідентифікаторами зчитувати оперативні дані з відповідних датчиків.

З огляду на вищезазначене, потоки даних у сценарії виконання аналітичного запиту виглядають, як показано на діаграмі.

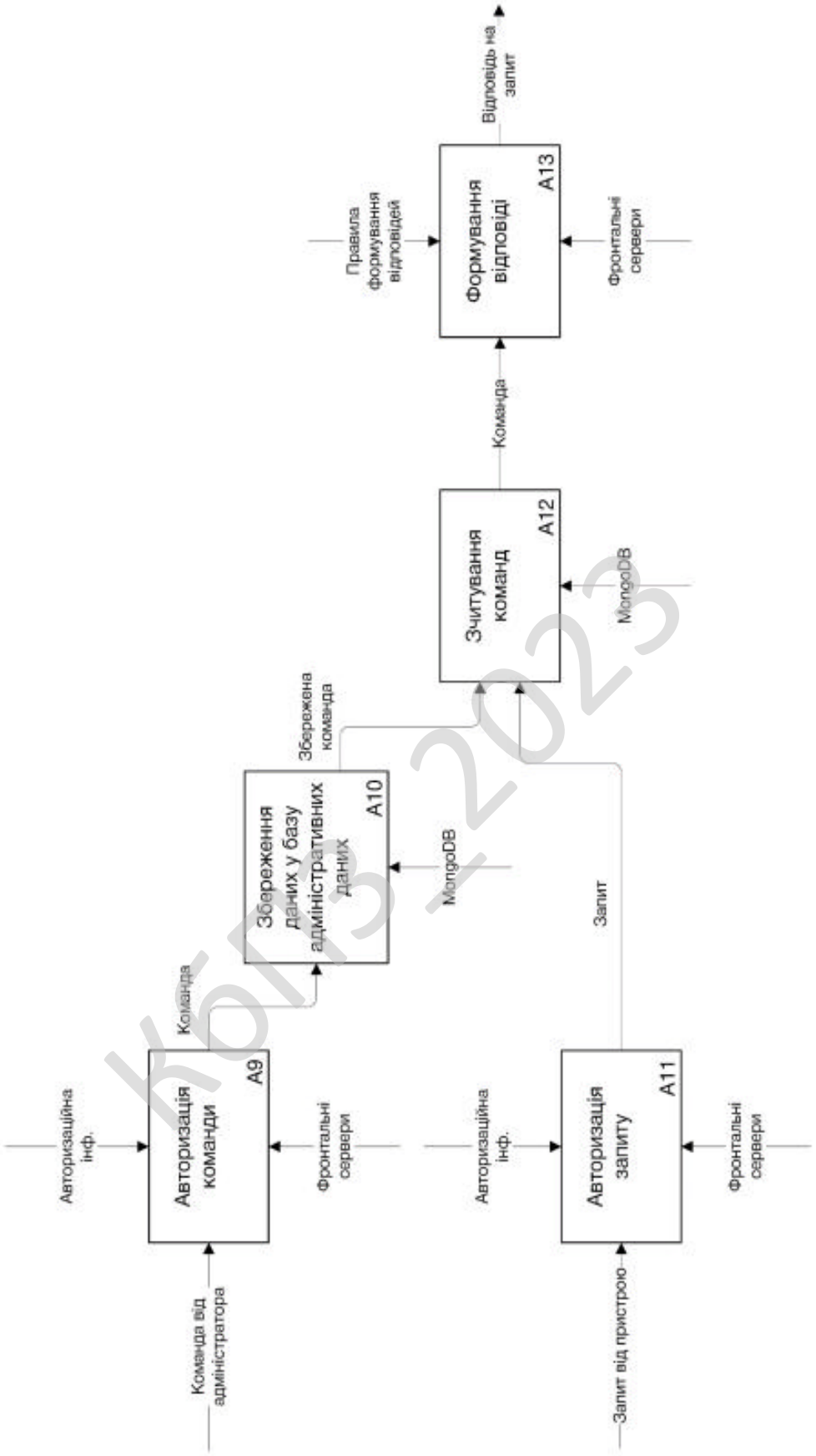

Рисунок 4.2 – Діаграма потоків даних і процесів при передачі команд від адміністратора до пристрою

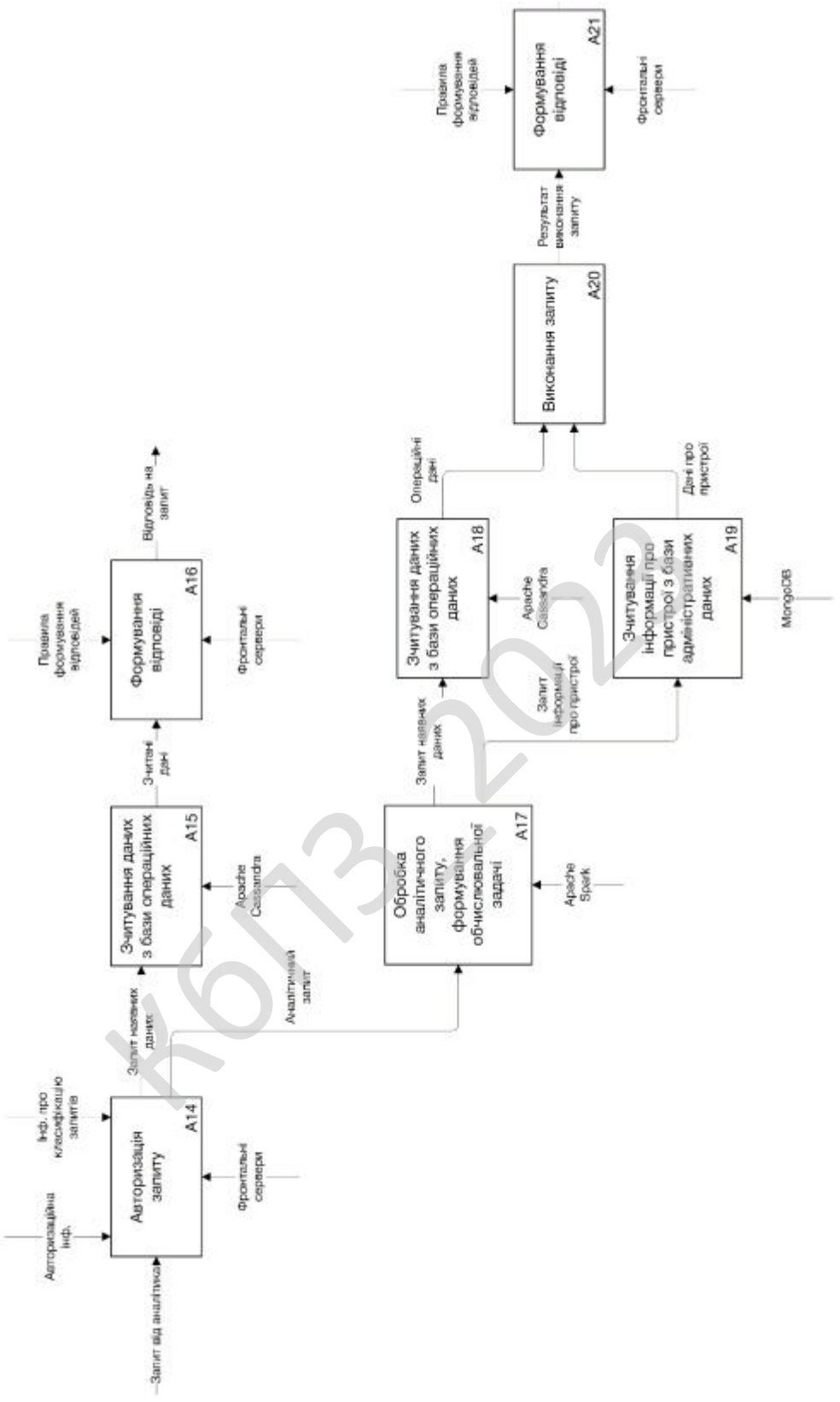

Рисунок 4.3 – Діаграма потоків даних і процесів при обробці запитів від аналітика

Протоколи та формати обміну даними між компонентами системи Опишемо протоколи передачі даних між компонентами системи. Зв’язок між пристроями (або шлюзами) і платформою відбувається через зашифрований протокол HTTP/2, який використовує криптографічний протокол TLS 1.2. Шифрування трафіку потрібне, щоб уникнути перехоплення даних та інших атак. Протокол HTTP/2, стандартизований [17] у 2015 році, дуже підходить для IoT, оскільки:

- розроблено як оптимізацію HTTP/1.1;

- реалізує стиснення заголовка HTTP (HPACK) за допомогою коду Хаффмана. Цей спосіб високоефективний, але вимагає мало пам'яті, що дуже важливо для малопотужних пристроїв;

- за допомогою таких прийомів, як конвеєрність і мультиплексування, стимулює використання єдиного TCP-з'єднання, що дозволяє уникнути повільних тристоронніх рукостискань, які, крім того, вимагають додаткових ресурсів. У зашифрованому протоколі повторне використання з’єднання дозволяє уникнути ще більш тривалого діалогу TLS (узгодження);

- забезпечує виконання операції PING на рівні протоколу, що дозволяє з її допомогою перевіряти правильність з'єднання з пристроєм;

- підтримує високорівневу сумісність з HTTP/1.1: методи, коди стану, URI, поля заголовків. Багато платформ IoT використовують MKTT, який набагато ефективніший, ніж HTTP/1.1. Проте з появою HTTP/2 HTTP/2 і MKTT стали майже рівними за продуктивністю.

Основною перевагою використання HTTP є його поширеність і загальне визнання в Інтернеті, що означає наявність функціональних і стабільних серверних і клієнтських бібліотек.

HTTP/2 реалізує техніку серверного проштовхування, яка дозволяє серверу ініціювати надсилання даних клієнту замість традиційної моделі запит-відповідь. Однак більш природним і ефективним для повного дуплексного зв'язку є використання протоколу WebSocket [18]. Пристрої, якщо вони можуть

встановити з’єднання з сервером, підтримують відкритий канал, через який вони можуть отримувати адміністративні команди від сервера за допомогою протоколу WebSocket. Використовується захищена версія протоколу WebSocket - WebSocket Secure (VSS). JSON використовується як основний формат даних, у якому зовнішні сервери отримують інформацію. Він, як і HTTP, дуже поширений; читається людиною (тобто не двійковий); простий у створенні та аналізі. Недоліком порівняно з двійковими форматами є розмір повідомлення, але його можна подолати за допомогою стандартних методів стиснення, таких як gzip. Загалом зовнішні сервери реалізують REST API, який використовується пристроями, адміністраторами та аналітиками. Як уже згадувалося, зв’язок між пристроєм і точкою входу в хмарну інфраструктуру платформи відбувається за допомогою зашифрованого з’єднання. Розшифровка (термінація SSL) відбувається на рівні балансувальника навантаження, а зовнішні сервери отримують уже відкриті дані. Це можна зробити, оскільки з’єднання всередині центру обробки даних можна вважати безпечним: трафік або ізольований від інших користувачів хостингу за допомогою засобів віртуалізації, або інфраструктура платформи може бути розташована в повністю фізично ізольованому сегменті мережі.

Крім того, він дозволяє виконувати дешифрування за допомогою спеціально адаптованих більш продуктивних рішень, витрачаючи при цьому ресурси фронтенд-серверів тільки на базову логіку обробки запитів. Передача даних між чергою повідомлень, підсистемою обробки даних, оперативною базою даних, адміністративною базою даних і зовнішнім сервером відбувається в певних двійкових форматах через TCP, реалізованих відповідними бібліотеками. Наприклад, підсистема обробки даних може обмінюватися серіалізованими об’єктами між своїми робочими серверами під час обчислень. Наочно показані формати даних і протоколи їх передачі між різними компонентами системи рисунку 4.4.

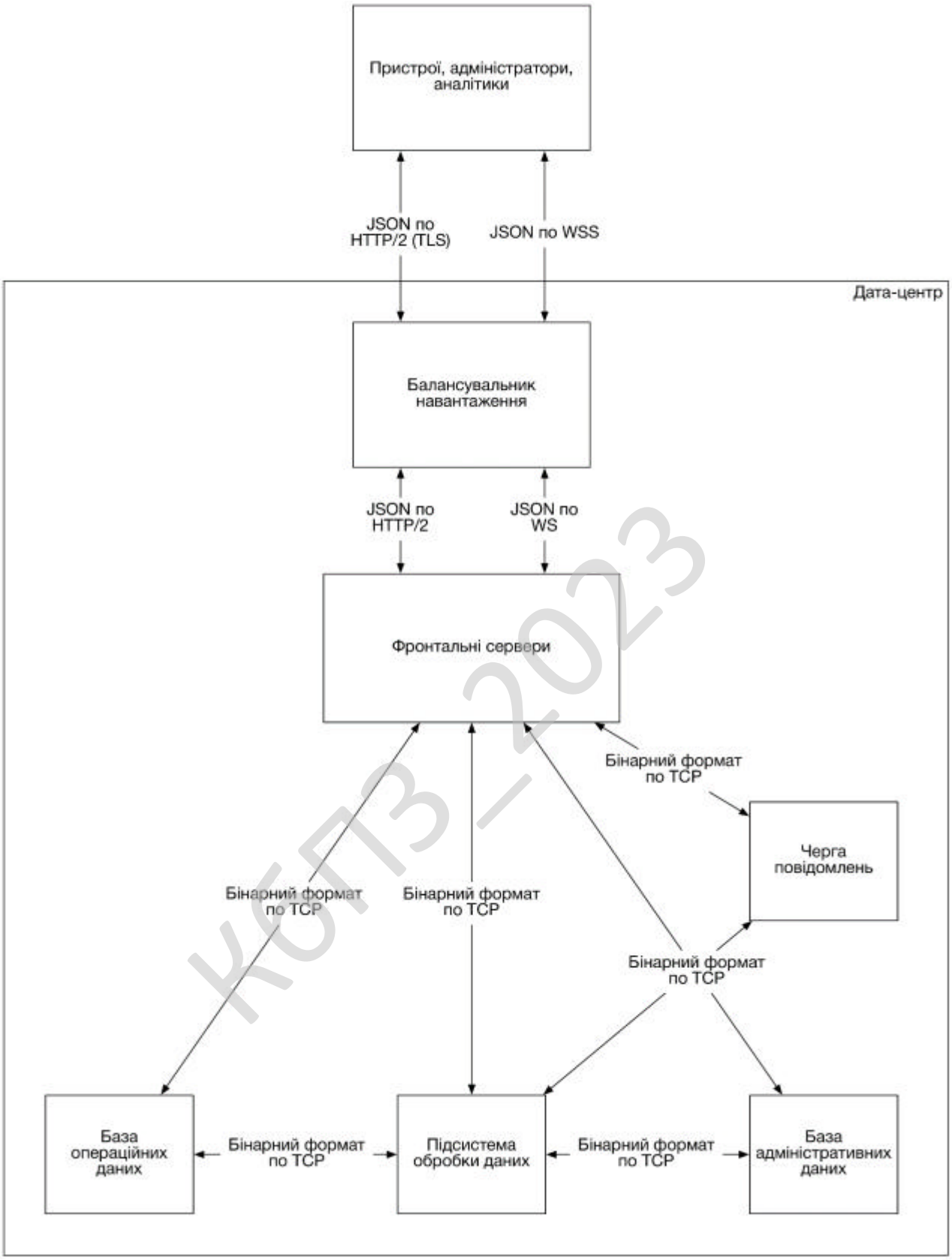

Рисунок 4.4 – Запропоновані формати і протоколи їх передачі між компонентами системи

Мікросервісна архітектура

Стиль архітектури мікросервісу - це підхід до розробки повної програми як набору невеликих служб, кожна з яких працює у своєму власному процесі та підключається до інших за допомогою легких механізмів, таких як RPC або HTTP. Сервіси будуються під конкретне завдання і можуть розгортатися самостійно за допомогою автоматизованих систем. Існує мінімальне централізоване управління такими послугами. Традиційні серверні системи зазвичай будуються як моноліти – логічно окремі виконувані файли. І такий підхід є природним: уся логіка обробки запитів виконується в одному процесі, що дозволяє використовувати наявні інструменти мови програмування, щоб розділити вашу програму на класи, функції та простори імен. Моноліт можна масштабувати горизонтально, запустивши багато екземплярів із балансувальника навантаження. Монолітне програмне забезпечення досить успішне, але воно має свої недоліки. Цикли зміни моноліту взаємопов'язані, тобто зміна невеликої частини системи вимагає перезбирання (компіляції) і розгортання. З часом стає важко підтримувати хорошу модульну структуру, зберігаючи зміни, що стосуються модуля, лише всередині цього модуля. Ви повинні масштабувати весь моноліт замість окремих частин, які вимагають більше ресурсів. Мікросервіси можна розгортати та масштабувати незалежно один від одного. Вони також забезпечують чіткі межі між модулями, навіть дозволяючи реалізовувати окремі підсистеми різними мовами програмування. Це розмежування допомагає керувати складністю кодової бази, оскільки кожен модуль матиме загальнодоступний API, який міститиме лише необхідні йому функції, а все інше буде інкапсульованим і не матиме відношення до розробки інших служб, які від нього залежать. Основним недоліком архітектури мікросервісів є підвищена складність обробки помилок. На відміну від моноліту, будь-який виклик служби може вийти з ладу. Тому необхідно розробити механізми моніторингу стану сервісів, перевірки різних метрик їх функціонування, а також автоматизувати відновлення мікросервісів у разі його збою. Позитивним є те, що збої в

архітектурі мікросервісів здебільшого поодинокі. Якщо для виконання запиту потрібно викликати багато служб, а деякі з них недоступні, можна дати менш повну відповідь (витончена деградація). Труднощі також можуть бути викликані забезпеченням послідовності розгортання залежних мікросервісів, а також необхідністю управління транзакціями, які взаємодіють з кількома підсистемами. Як бачимо, запропонована архітектура багато в чому увібрала ідеї мікросервісної архітектури. Компоненти чітко розділені та мають власний спеціальний набір функцій. Інтерфейсні сервери забезпечують єдину точку входу в систему. Коли запит отримано, відповідно до практики архітектури мікросервісу, зовнішні сервери здійснюють необхідну кількість викликів API служби, а потім об'єднують отримані результати для формування відповіді. Як описано, внутрішній зв'язок між серверами відбувається у двійковому форматі через TCP з міркувань ефективності, але з точки зору коду кожна служба надає загальнодоступний API, який інкапсулює створення повідомлень для відповідної служби. Окрім розподілу обов'язків, основною перевагою є можливість горизонтального масштабування кожної послуги окремо. Наприклад, якщо значно прискорити надходження оперативних даних без збільшення потоку адміністративних даних, відповідну СУБД можна масштабувати окремо.

Підсистема прийому даних. Організація, вимоги та обґрунтування вибору свого рішення

Оскільки дані надходять із високою швидкістю та можуть виникати стрибки трафіку, вам потрібен буфер між базою даних і зовнішнім сервером, щоб збалансувати навантаження. Для цього в архітектурі платформи передбачено використання черги повідомлень. Черги повідомлень визначають асинхронний протокол зв'язку. Це означає, що відправник і одержувач повідомлення не можуть спілкуватися з чергою повідомлень одночасно. Повідомлення в черзі зберігаються, доки їх не отримає одержувач. Важливою особливістю черги повідомлень є забезпечення стабільності та надійності, які реалізуються за допомогою різних стратегій зберігання даних. Для підвищення надійності

доставки повідомлень є можливість зберігати їх на диску до того, як їх отримає одержувач. Навіть якщо програма-одержувач або черга повідомлень виходить з ладу через збій, повідомлення будуть безпечними та доступними для одержувачів, щойно буде створено резервну копію системи. Використовуючи чергу повідомлень, ви можете підтримувати набагато більший обсяг повідомлень. Загалом, впровадження архітектури черги повідомлень є успішною стратегією для організації асинхронної обробки великих даних. Сформулюємо вимоги до черги повідомлень, які необхідно враховувати при виборі між доступними рішеннями:

1) Висока продуктивність. Необхідно забезпечити високу швидкість запису повідомлень у чергу.

2) Горизонтальна масштабованість. Можливість збільшення пропускної здатності системи за рахунок збільшення кількості серверів.

3) Можливість збереження повідомлень на диск. Необхідно зменшити кількість втрачених даних, коли сервер черги повідомлень вимкнено. Це особливо необхідно при зчитуванні даних з черги великими пакетами з великими інтервалами - в цьому випадку при відсутності постійного сховища велика кількість даних може бути втрачено.

4) Тиражування. Для певного збільшення обсягу зберігання даних він також гарантує доступність черги повідомлень для одержувачів у разі вимкнення частини сервера.

5) Можливість налаштування рівня гарантії надійності зберігання повідомлень. Тобто, на скільки серверів реплікується та як часто повідомлення зберігаються на диску. Необхідно налаштувати систему для більш надійного збереження повідомлень з пристроїв, які мають тривалий період між операціями надсилання даних. У разі великого навантаження слід мати можливість послабити такі гарантії, навпаки.

6) Можливість розподілу повідомлень на групи за темами. Таким чином, можна дати вказівку одержувачам повідомлень, які реалізують логіку, специфічну

для певного типу пристрою, отримувати повідомлення з певної черги, в якій розміщені відповідні дані. Apache Kafka найкраще задовольняє описані потреби. Обґрунтуємо цей вибір.

Apache Kafka – черга повідомлень для прийому даних

Apache Kafka — це брокер повідомлень з відкритим кодом, розроблений Apache Software Foundation і написаний на Scala. Kafka — це розподілений, розділений (розділений), реплікований журнал записів (журнал фіксації). Він пропонує функції обміну повідомленнями, але має унікальну архітектуру [20]. Спочатку коротко опишемо термінологію:

- стрічка повідомлень розділена на категорії, які називаються темами;

- процеси, які публікують повідомлення в Kafka, називаються виробниками;

- процеси, які підписуються на теми та обробляють потік опублікованих повідомлень, називаються споживачами.

- kafka працює як кластер, що охоплює один або більше серверів, кожен з яких називається брокером. Kafka копіює журнали для кожного розділу теми на налаштовану кількість серверів (цей параметр можна встановити для кожної теми окремо). Це дозволяє автоматично переключатися на репліку в разі збою сервера в кластері, щоб повідомлення залишалися доступними навіть за наявності збоїв. Kafka широко використовує файлову систему для зберігання та кешування повідомлень. Хоча диски зазвичай вважаються повільними, правильно спроектована дискова структура може працювати з тією ж швидкістю, що й мережа.

Лінійне читання та запис є найбільш передбачуваними моделями використання, тому операційна система їх ефективно оптимізує. Сучасні операційні системи забезпечують методи випереджального читання (читання сторінки кешу) і запису позаду (запис сторінки кешу), які попередньо завантажують дані у великі блоки та групують малі логічні операції запису у великі фізичні операції. Щоб компенсувати розрив у продуктивності між

оперативною пам’яттю та дисковою пам’яттю, сучасні операційні системи дедалі агресивніше використовують основну пам’ять для кешування диска. Майже всю вільну пам'ять можна використовувати для кешування диска з мінімальними витратами на звільнення. Усі операції читання та запису проходять через цей єдиний кеш. Таку функцію неможливо легко вимкнути без використання прямого вводу-виводу, тому, якщо процес підтримує власний кеш, дані, швидше за все, дублюються в кеші сторінок ОС, тобто одна й та сама інформація зберігається двічі. Крім того, оскільки Kafka працює на JVM, тоді:

- витрати на зберігання об'єктів дуже високі, що часто подвоює розмір даних (або більше);

- Збирання сміття JVM значно сповільнюється, коли розмір даних, що зберігаються в купі, збільшується.

Враховуючи наведені вище фактори, використання файлової системи з кешуванням сторінок є більш ефективним, ніж підтримка кешу чи іншої структури в пам’яті, оскільки вона: принаймні подвоює доступний кеш шляхом автоматичного доступу до всієї вільної пам’яті, і, швидше за все, вдвічі більше завдяки зберіганню компактна структура байтів замість окремих об'єктів. У результаті розмір кешу може досягати 28-30 ГБ на машині з 32 ГБ пам’яті без накладних витрат на збирання сміття. Крім того, цей кеш зберігатиме дані навіть після перезапуску процесу, тоді як внутрішній кеш потрібно перебудувати (що може зайняти до 10 хвилин для 10 ГБ кешу), інакше продуктивність буде дуже низькою, якщо почати з порожнім. .

Це також спрощує реалізацію, оскільки підтримка узгодженості між кеш-пам’яттю та файловою системою обробляється ОС, яка робить це більш ефективно та коректно порівняно з реалізаціями в процесі.

Якщо модель використання диска показує часте лінійне читання, тоді читання наперед заповнює кеш необхідними даними для кожної операції читання. Натомість, щоб зберегти кеш якомога більшим і швидко скинути дані на диск, коли місце закінчиться, робиться навпаки. Усі дані негайно записуються у

файлову систему, але це не обов'язково спричиняє операції фізичного запису. Це означає, що дані переміщуються на сторінки кешу ядра. Пакетна обробка - гарантія ефективності.

Щоб забезпечити це, постачальник даних (виробник) намагається накопичувати дані в пам'яті та надсилати більші пакети за одну операцію. Пакетну обробку можна налаштувати так, щоб накопичувати не більше певної фіксованої кількості повідомлень або очікувати не більше певного ліміту затримки (скажімо, 10 мс). Це дозволяє виконувати менше операцій введення-виведення, кожна з яких матиме більший обсяг. Настроювана буферизація дозволяє знайти компроміс між додатковою затримкою та вищою пропускною здатністю.

Споживач даних (споживач) працює, надсилаючи запити на вибірку брокерам, які керують розділами, з яких йому потрібні дані. З кожним запитом він визначає зміщення в журналі та отримує сегмент журналу, починаючи з указаної позиції. Таким чином, споживач даних має значний контроль над цією позицією і тому може перечитати повідомлення якщо потрібно.

Можна виділити декілька різних видів доправлення повідомлень:

- максимум один раз (не більше одного разу) – повідомлення можуть бути втрачені, але більше ніколи не доставлені;

- принаймні один раз – повідомлення ніколи не втрачаються, але можуть бути відправлені повторно;

- рівно один раз – повідомлення доставляється рівно один раз.

Ми опишемо семантику доставки повідомлень у Kafka. При публікації повідомлення існує поняття запису (внесення) повідомлення в журнал. Після написання повідомлення воно не буде втрачено, доки принаймні один посередник, який обслуговує розділ, до якого воно було написано, залишається в робочому стані. Якщо постачальник даних відчуває помилку мережі під час публікації повідомлення, він не може визначити, чи сталася помилка до чи після написання повідомлення. Ця семантика подібна до вставки рядка в таблицю в

базі даних із автоматично згенерованим ключем. Таким чином, Kafka за замовчуванням гарантує доставку «принаймні один раз» і дозволяє користувачеві реалізувати схему «принаймні один раз», вимикаючи повторні спроби на стороні постачальника та зміщення запису перед обробкою пакетів повідомлень. Семантика «точно один раз» вимагає співпраці з кінцевою системою зберігання даних.

Apache Kafka використовує Apache Zookeeper, розподілену службу конфігурації та синхронізації. Він використовується для координації вузлів у кластері, моніторингу статусу брокера, запису позицій під час читання повідомлень тощо.

Підсистема обробки даних

Архітектура Lambda — це архітектура обробки даних, розроблена для роботи з великими обсягами даних, одночасно використовуючи переваги пакетного та потокового підходів до обробки даних. Цей архітектурний підхід намагається збалансувати затримку, пропускну здатність і відмовостійкість за допомогою пакетної обробки, щоб забезпечити всебічне й точне представлення пакетів даних, одночасно використовуючи потокове передавання в реальному часі для створення робочих фрагментів даних. Ці дві частини даних можна об’єднати перед виведенням. Поширення лямбда-архітектур зумовлене збільшенням обсягів даних, потребами в аналітиці в реальному часі та зусиллями щодо зменшення затримок MapReduce. Архітектура Lambda працює з моделлю даних, яка має незмінне джерело (лише додавання). Ця модель призначена для захоплення та обробки подій із мітками часу, які додаються до наявних даних, а не перезаписуються. Стан системи визначається природним упорядкуванням даних у часі. Архітектура Lambda показана на схемі (рисунок 4.5).

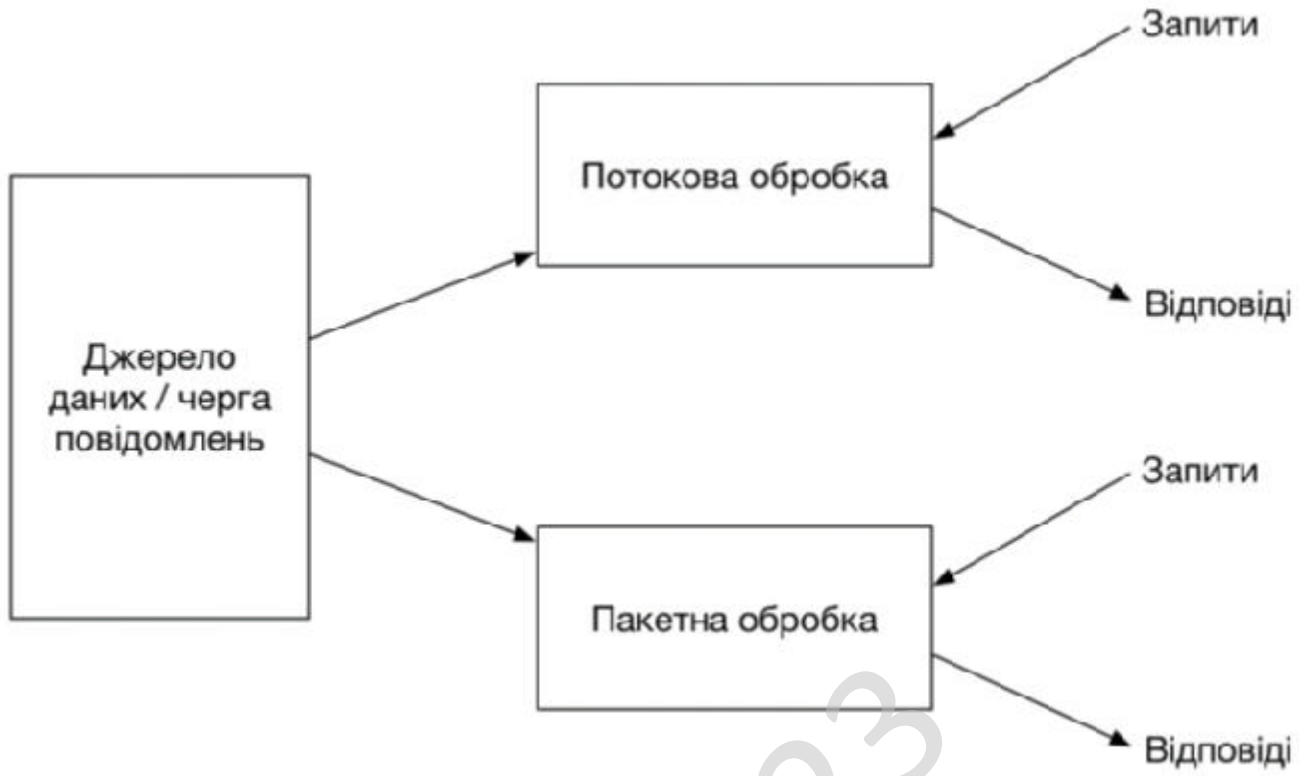

Рисунок 4.5 – Лямбда-архітектура

Також опишемо підхід MapReduce, який частково використовується для обробки даних платформою. Цей підхід призначений для обробки завдань, які є паралельними. Він складається з наступних кроків:

- Карта: кожен робочий сервер застосовує функцію карти до даних і записує результат у тимчасову пам’ять. Головний сервер забезпечує обробку лише однієї копії вхідних даних.

- Shuffle: робочі сервери розподіляють дані відповідно до вихідних ключів (виходи функції карти), щоб усі дані, що відповідають одному ключу, були на одному робочому сервері.

- Скорочення: робочі сервери обробляють кожну групу даних, відповідно до ключа, паралельно.

Розглянемо обчислення в парадигмі MapReduce на прикладі обробки значень температури, отриманих від декількох пристроїв.

Обробка буде складатися з усереднення значень температури для кожного пристрою (це робиться протягом певного періоду часу). На вході ми маємо

записи JSON, які містять ідентифікатор пристрою (всього їх 3) в полі «deviceId» і значення температури в полі «temp». Початкова фаза обробки (карта) формує пари ключ-значення: ключ – це ідентифікатор пристрою, а значення – температура. Вважайте, що MapReduce працює на трьох серверах. На етапі «тасування» дані розподіляються так, що перший робочий сервер містить дані з першого пристрою, другий — з другого і так далі. Після «тасування» на кожному сервері залишаються записи з однаковими значеннями ключів. Етап «зниження» полягає в обчисленні середнього арифметичного значень температури. Наприкінці розрахунку ми маємо три пари «ключ-значення», що містять, відповідно, ID пристрою та середнє значення температури, отримане з нього.

Apache Spark – система обробки даних

Apache Spark — це система обробки великих даних з відкритим кодом, створена для високої швидкості, простоти використання та складної аналітики [21]. Розроблено в 2009 році в AMLab (UC Berkeley), публічний випуск як проект Apache відбувся в 2010 році. Spark має значні переваги перед іншими технологіями Big Data і MapReduce, такими як Hadoop і Storm. Перш за все, Spark забезпечує комплексну та уніфіковану структуру для роботи з наборами даних різного характеру (текст, графік тощо) та джерела (пакет або потік).

Було доведено, що Spark [22] працює до 100 разів швидше, ніж Hadoop, під час виконання операцій у пам’яті та до 10 разів швидше, якщо використовується диск. Він має велику бібліотеку вбудованих даних і алгоритмів. Окрім операцій зіставлення та скорочення, він підтримує запити SQL, потокову передачу, машинне навчання та обробку даних графіків. Ці можливості можна використовувати окремо або об'єднати в єдиний ланцюжок обробки даних. Hadoop і Spark. Hadoop як технологія обробки великих даних популярна вже близько 10 років і довела свою ефективність. MapReduce є чудовим рішенням для однопрохідних обчислень, але не настільки ефективним для завдань, які потребують багатопрохідних обчислень і алгоритмів. Кожен крок у ланцюжку обробки даних має одну фазу Map і Reduce, і вирішення будь-якої проблеми має

бути описано цими операціями. Вихідні дані кожного кроку повинні бути збережені в розподілену файлову систему перед початком наступного кроку. Таким чином, цей підхід є досить повільним через реплікацію та зберігання на диску. Крім того, рішення на основі Hadoop зазвичай використовують складні для створення та керування кластери. Їм також потрібно інтегрувати кілька інструментів для різних випадків використання (наприклад, Mahout для машинного навчання та Storm для потокового передавання). Для складних обчислень кілька завдань MapReduce мають бути об’єднані та виконуватися послідовно. При цьому кожне завдання має тривалу затримку і не може початися до повного завершення попереднього. Spark дозволяє розробляти складні багатоетапні ланцюжки обробки даних за допомогою шаблону спрямованого ациклічного графіка (DAG). Він також підтримує обмін даними в пам’яті за допомогою різних графіків, що дозволяє виконувати кілька завдань над одними даними. Spark працює з існуючою інфраструктурою розподіленої файлової системи Hadoop (HDFS), розширюючи та додаючи до неї функціональність. Особливості Spark. Spark покращує MapReduce, забезпечуючи швидший і ефективніший етап Shuffle під час обробки даних. Продуктивність у декілька разів вища, ніж у інших технологій великих даних, завдяки зберіганню даних в оперативній пам’яті та обробці майже в реальному часі. Spark підтримує відкладене оцінювання запитів, що допомагає оптимізувати їх виконання. API високого рівня спрощує розробку та забезпечує узгоджену архітектурну модель для складних рішень. Проміжні результати зберігаються в пам’яті, а не на диску, що корисно, коли над певним набором даних виконується кілька операцій. Зазвичай система обробки працює як у пам'яті, так і на диску. Якщо дані не поміщаються в пам'ять, виконуються зовнішні операції. Spark можна використовувати для обробки наборів даних, розмір яких перевищує загальну пам’ять серверів у кластері. Spark намагається зберегти якомога більше даних у пам’яті. Деякі дані можуть бути на диску. Використання оперативної пам’яті є однією з головних переваг у продуктивності, але ви повинні відповідно

проаналізувати вимоги до обладнання, щоб використовувати її ефективно. Стійкий розподілений набір даних [23] (RDD) є основною концепцією Spark. RDD можна порівняти з таблицею в базі даних. Він може містити будь-які типи даних. Spark зберігає RDD на різних розділах. Ця абстракція дозволяє оптимізувати обробку шляхом зміни порядку обчислень. Вони також відмовостійкі, оскільки RDD знає, як відтворити або перерахувати набір даних. RDD є незмінними. Його можна змінити шляхом перетворення, але він повертає новий RDD, а вихідний RDD залишається незмінним. Підтримуються два типи операцій:

- Перетворення. Повертає новий RDD. Виклик функції перетворення RDD повертає новий RDD, вони не запускаються, доки обчислення не розпочато явно (ліниві обчислення). Прикладами перетворень є map, filter, flatMap, groupBiKey, reduceBiKey, aggregateBiKey, pipe та coalesce.

- Дії. Дія виконується негайно та повертає значення. Приклади: зменшити, зібрати, підрахувати, спочатку, взяти, countBiKey і foreach.

Окрім API Spark Core, існують додаткові бібліотеки, які є частиною екосистеми Spark і надають додаткові можливості в сферах аналітики та машинного навчання. Ці компоненти включають:

- Spark Streaming. Його можна використовувати для обробки потокових даних у реальному часі. Він заснований на мікропакетному стилі обчислень і обробки. Використовується абстракція DStream, яка є масивом RDD. Основна перевага полягає в тому, що той самий код, який описує ланцюжок обробки даних, можна використовувати як для пакетної, так і для потокової обробки даних.

- Spark SQL. Він надає можливість відкривати набори даних Spark через API JDBC і запускати SQL-запити за допомогою традиційних інструментів BI (бізнес-аналітики) і візуалізацій. За допомогою цієї бібліотеки ви можете виконувати ETL (Extract, Transform, Load) дані з різних форматів (JSON, Parquet,

реляційна БД та інші), перетворювати їх і робити доступними для спеціальних запитів.

- Spark MLlib. Масштабована бібліотека машинного навчання, що містить традиційні алгоритми навчання та утиліти, включаючи класифікацію, регресію, кластеризацію, спільну фільтрацію, зменшення розмірності та примітиви оптимізації низького рівня.

Операційна підсистема зберігання даних. Організація, вимоги та обґрунтування вибору стороннього рішення

Після первинної обробки дані записуються в оперативну базу даних, що забезпечує їх постійне зберігання. З цього сховища дані зчитуються як зовнішніми серверами для виконання запитів, так і підсистемою обробки даних для подальшого виконання обчислень, необхідних для задоволення аналітичного запиту. Отже, сформулюємо вимоги, яким має відповідати оперативна база даних:

1) Оптимізовано для запису. Зазвичай майже у всіх подібних сховищах оперативних даних у системах великих даних, як-от розглянута платформа, спостерігається головна особливість: кількість операцій запису значно перевищує кількість операцій читання. Цей шаблон використання відрізняється від того, для якого оптимізовані традиційні реляційні бази даних. Вони найкраще підходять для сценарію, коли дані лише час від часу записуються в базу даних, але запити на читання є частими. Наприклад, якщо є певна таблиця, яка містить новини на певному веб-сайті, очевидно, що записів набагато менше, ніж читань. Якщо сховище даних обслуговує систему IoT, воно повинно виконувати десятки тисяч запитів на запис у кожну секунду. Читання має здійснюватися лише за запитом адміністратора або аналітика. Також було б непогано мати високу продуктивність читання, але це набагато менш важливо, ніж швидкість запису.

2) Горизонтальна масштабованість. Збільшення кількості серверів має збільшити пропускну здатність для операцій запису, а також обсяг збережених даних.

3) Доступність. За допомогою реплікації має бути забезпечена доступність бази даних (для запису та читання) у разі відключення частини кластера.

4) Послаблені гарантії транзакцій. Оперативна база даних вимагає набагато слабших транзакційних гарантій, ніж традиційний набір ACID, гарантований реляційними даними. Наприклад, якщо ми говоримо про узгодженість, то більшість сценаріїв використання задовольнять граничну узгодженість.

5) Мінімальний набір функціональних можливостей для запитів. Оскільки підсистема обробки даних використовується для виконання запитів, оперативна база даних повинна забезпечувати лише мінімальний набір операцій, таких як запис, читання, видалення тощо.

Наприклад, якщо до запиту необхідно застосувати функцію агрегації (таку як сума, мінімум або максимум), то це зробить підсистема обробки даних, а операційній базі даних потрібно лише прочитати необхідні дані (відфільтровані за певним ключ, наприклад дані на певну дату).

Враховуючи вищезазначені вимоги, зрозуміло, що реляційна база даних навряд чи підійде для зберігання оперативних даних і що слід використовувати рішення NoSQL.

Реляційні бази даних записують дані набагато повільніше, ніж читають їх. Реплікація доступна в деяких базах даних SQL, але її функціональність часто обмежена та має певні характеристики через вимоги транзакцій ACID.

Горизонтальна масштабованість у реляційних базах даних (шардинг) дуже складна з тих же причин. Це призводить до того, що коли реляційній базі даних не вистачає дискового простору або вона не може обслуговувати необхідну кількість запитів за одиницю часу, можна лише збільшити потужність одного сервера (вертикальне масштабування), не витрачаючи багато часу та збільшуючи складність системи.

Крім того, необхідно зберігати великі обсяги даних, а продуктивність запитів до таблиць у реляційних базах даних значно знижується зі збільшенням

розміру таблиць. Для зберігання оперативних даних ми виберемо Apache Cassandra з баз даних NoSQL. Розповімо, як він влаштований і чому відповідає поставленим вимогам.

Apache Cassandra – база для зберігання операційних даних

Apache Cassandra — це система керування розподіленою базою даних із відкритим вихідним кодом, призначена для обробки великих обсягів даних, розподілених між великою кількістю неспеціалізованих серверів, і забезпечення високої доступності без єдиної точки відмови [24]. Cassandra надійна навіть для кластерів, частини яких розташовані в різних дата-центрах. База даних Cassandra написана на Java та включає повністю розподілену хеш-систему Dynamo, яка забезпечує майже лінійну масштабованість із збільшенням обсягу даних.

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Теорема CAP. Щоб зрозуміти Кассандру, варто спочатку згадати теорему CAP. Віна стверджує, що розподілена система не може надати більше ніж дві з наступних гарантій:

- узгодженість даних (всі вузли бачать однакові дані в будь-який час);

- доступність (гарантія отримання відповіді на кожен запит про його успішне або неправильне виконання);

- стійкість до поділу (незважаючи на поділ на ізольовані частини або втрату зв'язку з частиною вузлів, система не втрачає стабільності та здатності правильно реагувати на запити). Cassandra надає користувачам сильні гарантії доступності та відмовостійкості розділів, тобто, з точки зору теореми SAC, це AR-система. Послідовність налаштовується, що дозволяє знайти компроміс між узгодженістю та затримкою. Під час запису/читання даних ви можете встановити рівень узгодженості для кожного запиту. Модель даних BigTable / Log-structured.

У моделі даних BigTable первинний ключ і імена стовпців зіставляються з відповідними значеннями, утворюючи багатовимірне відображення. Кожен стіл

має багато вимірів. Мітка часу є одним із таких розмірів, який дозволяє керувати версіями таблиці, а також використовується для внутрішнього збирання сміття (видалених даних).

Таблиця 4.1 – Структура даних у Cassandra

Ключ рядка	Колонка 1	…	Колонка N
	Значення 1	…	Значення N
	Сімейство колонок		

Розглянемо приклад зберігання інформації для сценарію роботи з метеорологічними даними. У таблиці 4.2 ключ рядка є ідентифікатором пристрою («deviceId»), а значення температури і тиску записуються в стовпці «температура» і «тиск».

Таблиця 4.2 – Приклад зберігання даних

deviceId = 1	temperature	pressure
	24.3	738.4
deviceId = 2	temperature	pressure
	16.1	762.9

Варто відзначити, що в кожному рядку зберігаються не тільки значення, але і назви стовпців, що дозволяє структурі (схемі) бути динамічною. Родини колонок. Стовпці групуються в набори, які називаються сімействами стовпців, доступ до яких можна отримати за допомогою клавіші рядка. Створення сімейства стовпців є обов’язковим для зберігання даних. Зазвичай вони намагаються зберегти невелику кількість різних сімейств стовпців у просторі ключів, а сімейства змінюються лише час від часу. Однак кількість динаміків у родині необмежена. Ключовий простір. Ключовий простір — це група сімейств стовпців. Стратегії реплікації та контроль доступу реалізовані на рівні простору

ключів. У порівнянні з реляційними базами даних, ключовим простором є база даних (схема), а сімейством стовпців є таблиця. SSTable (Сортована таблиця рядків). SSTable — це формат зберігання файлів, який використовується Cassandra; це впорядкована незмінна структура рядків стовпців (пар "ім'я-значення"). Існує операція пошуку значення, пов’язаного з певним ключем, а також сортування пар ім’я стовпця – значення у вказаному діапазоні ключів. Кожна таблиця SSTable містить масив ключів рядка та набір пар ключ-значення стовпця. У файлі індексу є індекс і початкова позиція ключа рядка, які зберігаються окремо. Урізаний індекс завантажується в пам’ять, коли SSTable відкривається, щоб оптимізувати обсяг пам’яті, необхідний для індексу. Пошук рядків можна виконувати за допомогою пошуку на одному диску та послідовного сканування. Memtable. Memtable — це область пам’яті, до якої записуються дані під час операцій оновлення або видалення. Це тимчасове місце, з якого дані будуть записуватися на диск як SSTable, коли він заповнюється. Загалом операція оновлення або запису в Cassandra — це послідовний запис у журнал фіксації на диску та оновлення пам’яті; отже, запис відбувається так само швидко, як і запис у пам'ять. Виглядає так, як показано рисунку 4.6.

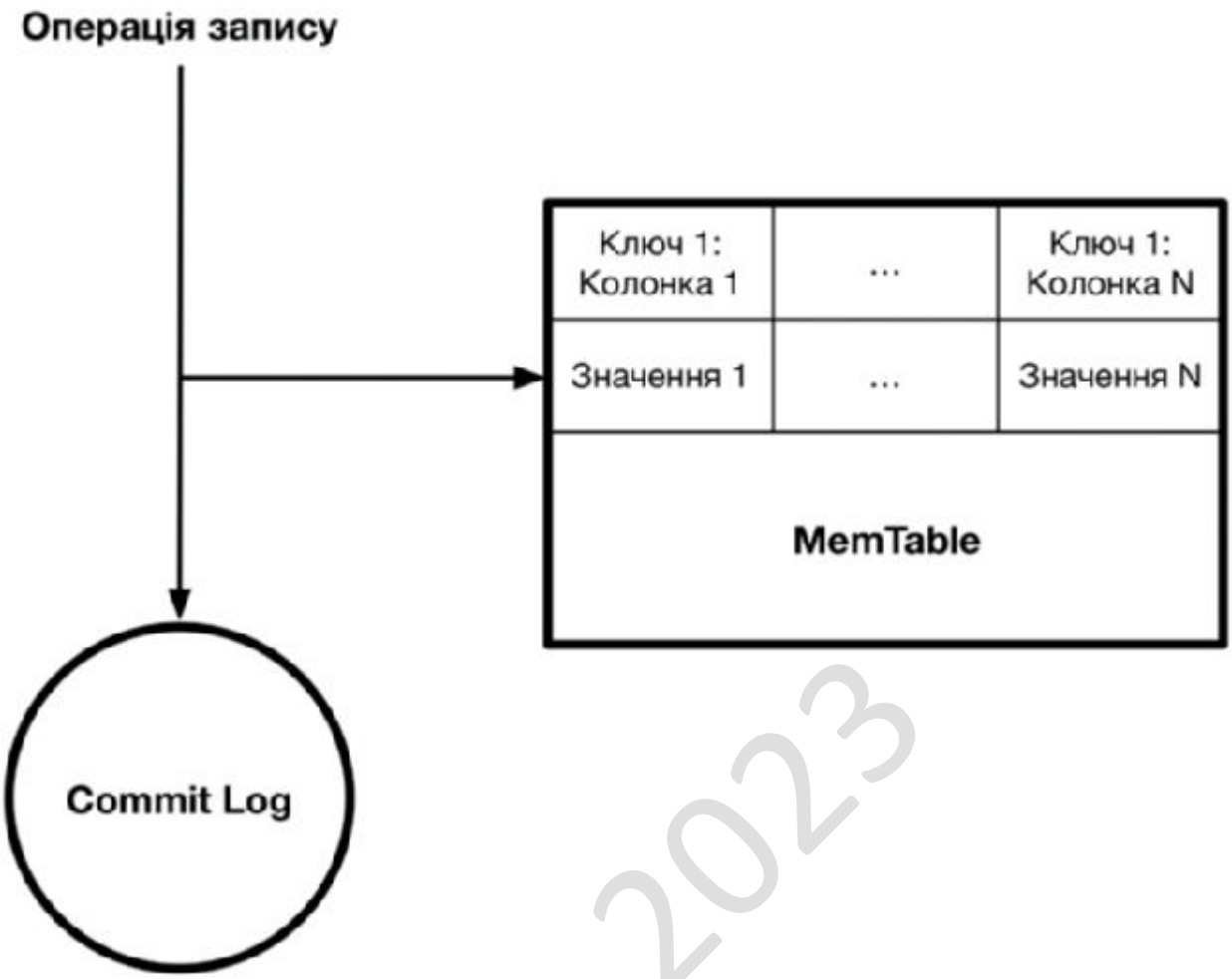

Рисунок 4.6 – Схема запису у Cassandra

Операції читання в Cassandra поєднують дані з різних SSTables і дані з Memtables. Запити на читання завжди повинні містити ключ рядка (первинний ключ), за винятком сканування діапазону. Коли надходить кілька запитів на оновлення певного стовпця, Cassandra використовує надані клієнтом позначки часу для вирішення конфліктів. Операції видалення працюють інакше: оскільки SSTables є незмінними, Cassandra записує так званий надгробний камінь, щоб уникнути випадкових записів. Надгробок — це спеціальне значення, яке записується замість миттєвого видалення даних і може бути видалено під час збирання сміття. Його можна надіслати вузлам у кластері, які не отримали перший запит на видалення. Ущільнення. Щоб обмежити кількість файлів, які беруть участь у читанні даних, і звільнити місце, зайняте непотрібними даними, Cassandra виконує стиснення. Таким чином, компактифікація поєднує кілька (це

число можна налаштувати) SSTables в одну велику SSTable. SSTables спочатку мають такий самий розмір, як Memtables. Таким чином, SSTtables експоненціально зростають у розмірі в міру старіння та надходження даних. Схема розбиття та реплікації подібна до описаної в статті Dynamo [25]. Протокол пліток. Cassandra — це однорангова система без єдиної точки відмови; інформація про топологію кластера передається за допомогою протоколу gossip. Протокол пліток схожий на плітки в соціальних комунікаціях (звідси і назва). Відповідно до цього протоколу вузол B передає іншим вузлам у кластері те, що він знає про стан вузла A. Ці вузли передають інформацію про A іншим, і через деякий час усі вузли дізнаються про стан A. Розподілена хеш-таблиця. Ключовою особливістю Cassandra є її здатність до поступового масштабування. Це включає в себе можливість динамічного розділення даних між набором вузлів у кластері. Cassandra поділяє дані на кластер за допомогою узгодженого хешування та рандомізує рядки в мережі відповідно до хешу ключа рядка. Коли вузол включається в кільце, йому призначається маркер, який вказує його розташування в кільці (рисунок. 4.7).

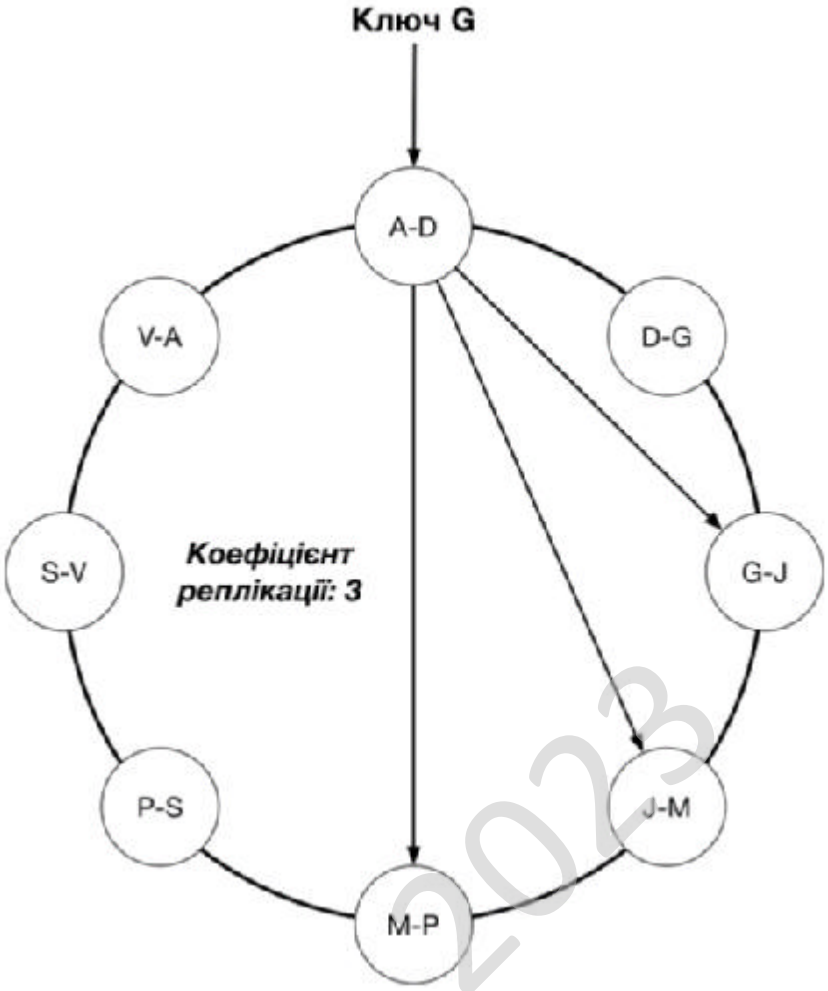

Рисунок 4.7 – Партиціонування даних при записі в Cassandra

Постійність підсумкового рахунку. Після достатнього періоду часу без змін будь-які оновлення мають поширюватися системою, синхронізуючи репліки. Cassandra підтримує як сильну узгодженість, так і остаточну узгодженість; це може контролювати клієнт, який виконує операцію. Cassandra підтримує різні рівні узгодженості під час запису та читання даних. Рівень узгодженості описує кількість реплік даних, до яких вузол-координатор повинен підключитися, перш ніж підтвердити клієнту завершення операції. Якщо V + R > K, де V — кількість реплік, які повинні підтверджувати записи, R — кількість вузлів, які повинні підтверджувати читання, K — коефіцієнт реплікації, тоді забезпечується надійна узгодженість. Доступні такі рівні консистенції:

- ONE: R/V принаймні для одного вузла;
- ALL: R/V для всіх вузлів для цього ключа розділу;

- КВОРУМ: R/V принаймні на ⌊K/2⌋+1 вузлах, де K — коефіцієнт реплікації.

Коли вузли не працюють на технічне обслуговування, Cassandra збереже інформацію про оновлення даних на цих вузлах для використання, щойно вузли знову стануть доступними. Крім того, для забезпечення узгодженості Cassandra використовує такі методи, як підказки щодо передачі даних, виправлення читання та виправлення антиентропії.

Програмна реалізація

Практична реалізація платформи виконана на Java (версія 8) та Scala (версія 2.11).

Програмний код основних компонентів наведено в додатку. Використовувалися такі бібліотеки:

- Undertow — це веб-сервер, написаний мовою Java, який дуже ефективний завдяки використанню неблокувального вводу-виводу. Це дозволяє описати логіку за допомогою окремих обробників, які обслуговують запити на певну адресу. За допомогою цієї бібліотеки реалізовані сервери переднього плану, тобто інтерфейси HTTP (REST API) і WebSocket;

- Kafka clients – клієнтські інтерфейси для взаємодії з кластером Apache Kafka. - Набір бібліотек та інтерфейсів для роботи з Apache Spark;

- Конектор Spark-Cassandra – драйвер Apache Cassandra з інтерфейсом, інтегрованим з API Apache Spark;

- MongoDB Async Driver – драйвер MongoDB, реалізований за допомогою Java NIO для неблокуючих запитів до бази даних;

- бібліотека для створення та перевірки веб-токенів JSON;

- Apache Commons Codec – набір реалізацій кодеків, наприклад, Base64 для формування веб-токенів JSON;

- SLF4J – Java API для запису різноманітних інформаційних та діагностичних повідомлень про хід програми;

- Logback – реалізація SLF4J;

- JUnit – бібліотека модульного тестування.

Структура бази адміністративних даних

Адміністративна база даних, побудована на MongoDB, має дві колекції: користувачів і пристроїв.

Колекція Користувачі зберігають інформацію про свої права доступу. Хоча MongoDB підтримує динамічну схему, базовий набір полів у документах є незмінним.

Ось структура документів цієї збірки:

{ "login": рядок,

"пароль": рядок,

"ролі": рядок,

"info": Об'єкт

}

Наведена вище структура описана в ієрархічному форматі, подібному до JSON, у якому ключі є іменами полів у документі, а значення — типами даних у цих полях.

Поля «логін» і «пароль» мають тип string і призначені для аутентифікації.

Поле «роль» визначає роль користувача і, відповідно, його права доступу. Наприклад, користувач може бути адміністратором, аналітиком тощо.

У реалізації Java доцільніше зберігати роль користувача як enum, але MongoDB це не підтримує.

Можна було б зберігати enum у MongoDB як число, але рядковий тип більш інтуїтивно зрозумілий, і той факт, що він менш компактний, не викликає проблем через малу кількість записів у цій таблиці. У полі «info» можна (опціонально) зберігати додаткову інформацію про користувача у вигляді об'єкта з довільними полями.

Колекція пристроїв містить інформацію про пристрої, які використовуються для їх автентифікації та виконання різноманітних запитів.

Структура документів цієї збірки така:

{ "device_id": рядок,

"пароль": рядок,

"type": рядок,

"status_messages": рядок

}

Поля «device_id» і «password» використовуються для автентифікації. "type" – тип пристрою, enum. "location" містить координати пристрою у форматі GeoJSON.

Поля «status_messages» і «commands» містять повідомлення про стан пристрою та команди відповідно. "info" - довільний об'єкт з додатковою інформацією.

Опис статусних повідомлень виглядає так:

{ "створено": позначка часу,

"bi": рядок, "types": рядок,

"корисне навантаження": об'єкт,

"отримано": позначка часу }

«created» і «received» — це, відповідно, мітки часу створення запису та його отримання адміністратором. "bi" - відправник повідомлення (підсистема або трекер). "type" - тип повідомлення (наприклад, інформаційне або повідомлення про помилку). "корисне навантаження" - це довільний об'єкт з описом вмісту повідомлення.

Об'єкти, що описують команди, зазвичай мають таку структуру:

{ "створено": позначка часу,

"корисне навантаження": об'єкт,

"отримано": позначка часу,

"результат": об'єкт }

Тобто містять дату і час створення (внесення до бази даних) – «створено»; відмітка часу моменту прийому пристроєм – «отримано»; об'єкт з довільним набором полів, що описують команду - "корисне навантаження"; і об'єкт з

динамічною структурою, який описує результат виконання пристроєм команди - “result”.

Розробка блок схем та алгоритмів роботи

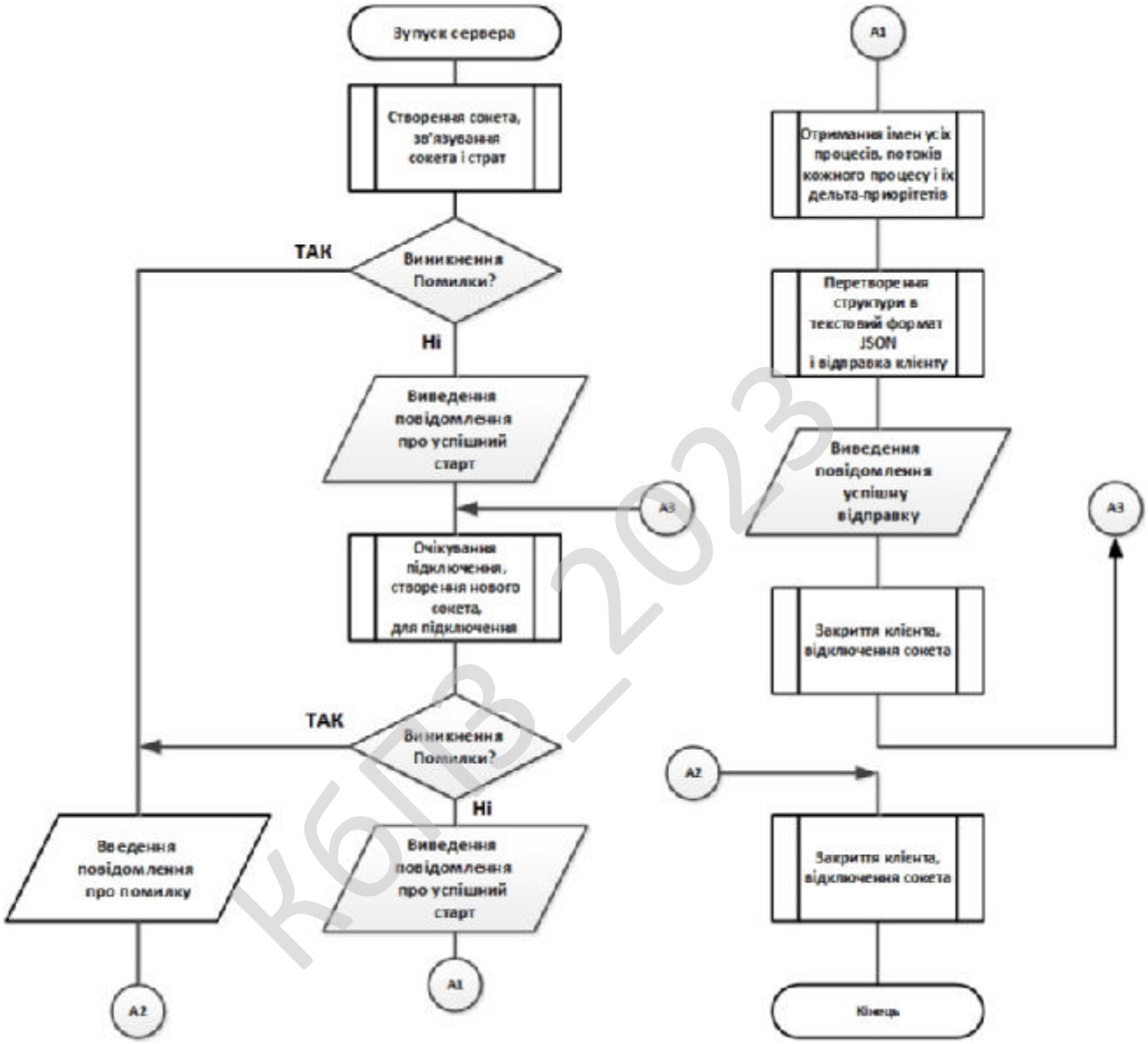

Рисунок 4.8 - блок-схему алгоритму підключення клієнта

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.9.

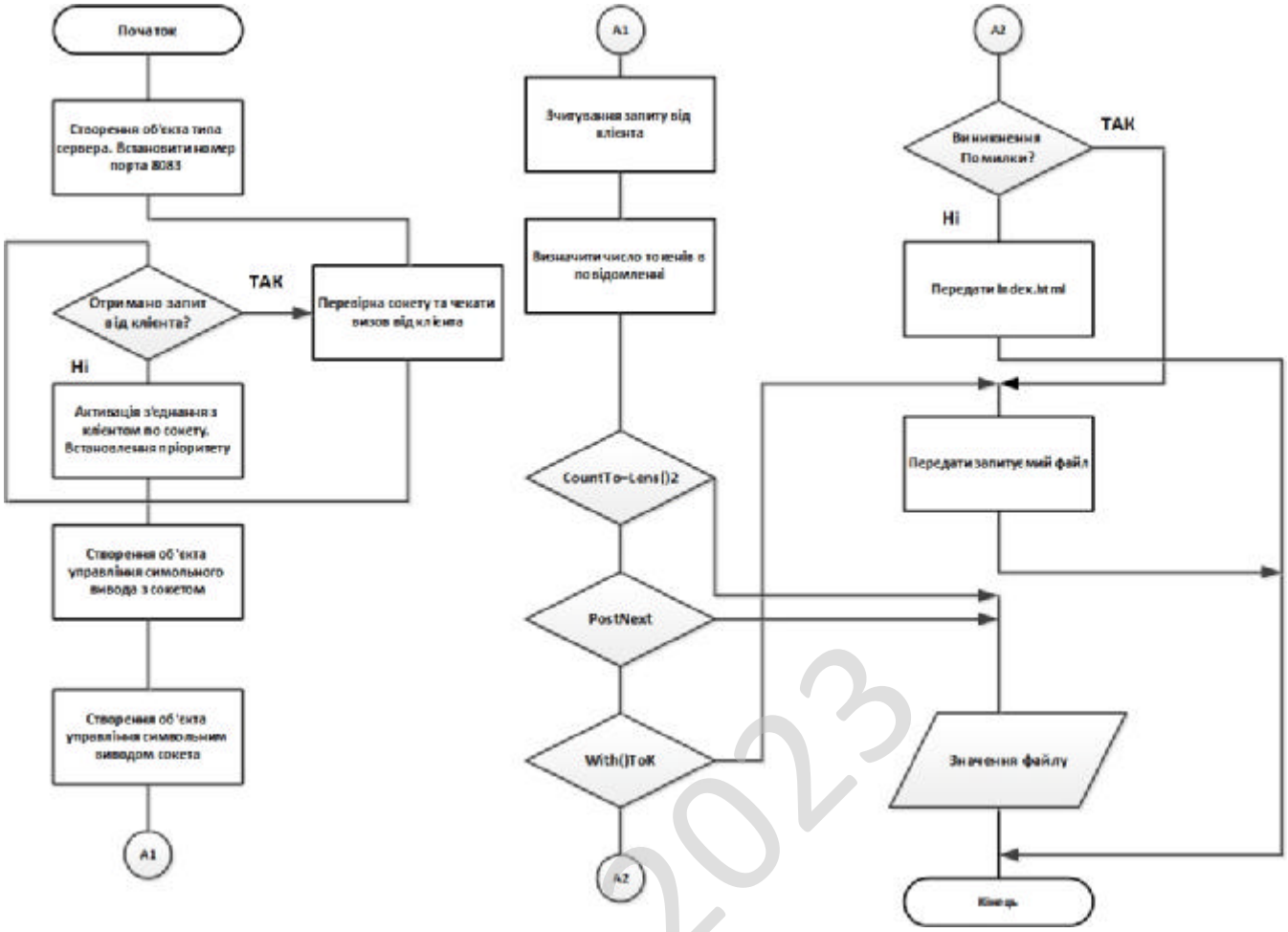

Рисунок 4.9 – Алгоритм роботи основної програми

4.2 Захист розробленого програмного забезпечення

Підсистема зберігання адміністративних даних. Організація, вимоги та обґрунтування вибору стороннього рішення для захисту інформації в ІБ

Платформа дозволяє передавати інформацію про стан від пристроїв до адміністраторів і команди від адміністраторів до пристроїв. Крім того, повідомлення про статус, призначені для адміністраторів, можуть генеруватися за допомогою тригерів відстеження. Як описано в розділі «Архітектура платформи», ці дані мають бути безпечно збережені перед передачею одержувачу.

Ця база даних також зберігає інформацію про автентифікацію/авторизацію (тобто права доступу користувача та пристрою) і дані пристрою (його тип, місцезнаходження та іншу інформацію).

Для безпеки паролі зберігаються в хешованій формі за допомогою алгоритму Bcrypt. Bcript — це функція хешування паролів, розроблена Нільсом Провосом і Девідом Мазьєром [26], яка базується на шифрі Blowfish.

На додаток до використання солі для захисту від атак веселкової таблиці, bscript є масштабованим: з часом кількість ітерацій можна збільшити, щоб сповільнити його обчислення, тому він залишається стійким до атак грубої сили, навіть коли кількість доступних обчислювальних ресурсів збільшується .

Ми опишемо вимоги до адміністративної бази даних:

1) Надійність зберігання. Якщо база даних підтверджує виконання запиту на запис, ми повинні бути впевнені, що дані збережені в постійному сховищі, тобто вони не будуть втрачені після збою сервера.

2) Сильна консистенція. Якщо адміністративна база даних є розподіленою, то після успішного запису певного значення операція читання повинна повернути те саме значення.

3) Тиражування. Мінімізуйте час простою бази даних у разі збою сервера, утримуючи головний сервер-репліку в режимі гарячого очікування та переключаючись на нього. Важливо для зберігання інформації про стан: пристрої можуть повторювати запити протягом деякого часу (якщо вони важливі), але їхня ємність пам’яті сильно обмежена.

4) Мова функціональних запитів. Сховище адміністративних даних містить інформацію про пристрої, які можна використовувати для виконання аналітичних запитів. Наприклад, вам може знадобитися відфільтрувати пристрої за певним критерієм. Підтримка геопросторових запитів також є обов’язковою. Оскільки платформа націлена на збір і аналіз даних з великої кількості просторово розподілених пристроїв, геопросторові запити необхідні для виконання аналітичних запитів, наприклад, пошуку пристроїв у певному радіусі

заданих координат. Повнотекстовий пошук (повнотекстовий пошук) також буде корисний при аналізі даних, що містяться в адміністративній базі даних. Наприклад, ви можете шукати повідомлення про статус, які містять певне слово.

5) Ефективні операції видалення. З описаних типів даних, що зберігаються в цій базі даних, найбільший обсяг мають повідомлення про стан і команди. Але більшість із них після отримання можна видалити. Тому база даних повинна підтримувати ефективні операції видалення, які є швидкими та звільняють місце на диску.

6) Динамічна структура даних. Зручно мати довільну структуру даних, щоб спростити адаптацію до мінливих форматів, які використовуються пристроями.

Реляційні бази даних загалом задовольняють вимогам (1)-(3). Вони традиційно забезпечують значні гарантії зберігання даних, а в нових версіях (наприклад, PostgreSQL 9.0+) мають підтримку режиму гарячого очікування.

Мова запитів SQL має широкі аналітичні можливості, але стандарт не описує геопросторові запити, тому вони доступні лише як розширення в деяких базах даних, а їх можливості дещо обмежені (наприклад, у MySQL [27]). Можливості повнотекстового пошуку також не дуже добре розвинені в реляційних базах даних, тому для цього потрібно використовувати окреме рішення.

Крім того, реляційні бази даних видаляють дані неефективно: такі операції дуже повільні, а звільнення місця на диску ще довше і складніше. Наприклад, звільнення місця в таблиці InnoDB (MiSKL) вимагає повторного створення таблиці (тобто створення нової таблиці, копіювання даних зі старої в нову та видалення старої) [27]. Більше ніж інші рішення, MongoDB відповідає нашим вимогам. Розглянемо, як це влаштовано і чому підходить.

MongoDB – база даних для зберігання адміністративних даних

MongoDB — це кросплатформна документоорієнтована база даних. Класифікований як NoSQL, MongoDB використовує динамічно структуровані

JSON-подібні документи (формат називається BSON) замість традиційних табличних структур реляційних баз даних, що робить інтеграцію даних швидшою. Розроблено MongoDB Inc. і випускається як безкоштовне програмне забезпечення з відкритим вихідним кодом під комбінацією GNU Affero General Public License та Apache License.

Станом на липень 2020 року MongoDB є четвертою за популярністю СУБД і найпопулярнішим документоорієнтованим сховищем. Давайте розглянемо основні функції MongoDB: Спеціальні запити. Підтримуються запити полів, діапазонів і регулярних виразів. Запити можуть читати окремі поля документа та включати визначені користувачем функції JavaScript, які виконуються безпосередньо сервером. Індексація. У документі можна індексувати будь-яке поле, включаючи масиви та вкладені документи (індекси в MongoDB концептуально подібні до індексів у реляційній СУБД). Доступні первинний і вторинний індекси. тиражування. Наявність підтримується наборами реплік. Набір реплік складається з двох або більше копій даних. Кожен такий пул може діяти як первинна або вторинна репліка. За замовчуванням основний виконує всі записи та читання, що означає, що забезпечується надійна узгодженість. Вторинні зберігають копію даних первинної репліки. Якщо основна репліка виходить з ладу, набір реплік автоматично вибирає, яка з вторинних реплік стане основною. Вторинні репліки можуть (необов'язково) виконувати операції читання, але дані будуть узгодженими в остаточному обліковому записі.

Балансування навантаження. MongoDB масштабується горизонтально за допомогою сегментування. Користувач вибирає ключ, який визначає, як дані розподіляються в колекції. Вони поділені на діапазони та розподілені по кількох серверах. Ключ шардингу також можна хешувати для рівномірного розподілу. Агрегація. Це дозволяє вам виконувати запити, подібні до SQL GROUP BY. Оператори агрегації можуть бути об'єднані разом як канали Unix. Також є пошуковий оператор, який об'єднує документи.

Швидкі операції видалення. MongoDB зберігає дані у подвійно зв'язаному списку, тому для видалення потрібно лише змінити два покажчики. Однак автоматичного стиснення немає, тому дисковий простір не звільняється автоматично, а реалізується спеціальною утилітою. Лімітовані колекції. Він підтримує колекції фіксованого розміру, які називаються обмеженими колекціями.

Вони підтримують порядок вставки, і коли заданий розмір досягається, вони поводяться як циклічна черга. Це може бути корисно для повідомлень про стан і команд, наприклад, щоб зберігати не більше ніж указану кількість записів. Використання таких колекцій може бути набагато ефективнішим, ніж регулярне видалення записів. Відсутність підтримки багатодокументних операцій. Транзакції ACID підтримуються лише на рівні документа. Але, як буде показано пізніше, структура документа, яка буде використовуватися, не вимагає багатодокументних операцій.

Динамічна структура. Оскільки значення в документах зберігаються разом з іменами полів, модель даних може змінюватися динамічно. Повнотекстовий пошук. Підтримується широкий спектр можливостей повнотекстового пошуку, для яких потрібен текстовий індекс.

Геопросторові запити. При наявності відповідного індексу можна виконувати різні геопросторові запити, наприклад, шукати точки на заданій відстані від заданої. При цьому обчислення ведуться з великою точністю, тому що відстань, наприклад, відраховується від точки на сфері, тобто враховується кривизна Землі.

Організація аутентифікації. Веб-маркер JSON

Автентифікація потрібна для запитів, що надходять від пристрою та інших користувачів системи (адміністраторів, аналітиків тощо). Необхідно забезпечити значну безпеку системи, яка, однак, дозволяє підтримувати високу пропускну здатність обробки запитів. Давайте виберемо JSON Web Token (JVT) - сучасний інструмент, який має багато переваг перед традиційними методами

аутентифікації. JVT — це відкритий стандарт [28], який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами у формі об’єкта JSON. Цю інформацію можна перевірити, оскільки вона має цифровий підпис. JVT можна підписувати за допомогою секретного слова (алгоритм HMAC) або пари публічно-приватних ключів (RSA). Через невеликий розмір JVT можна передати в URL-адресі, у параметрі POST або в заголовку HTTP. Крім того, компактність збільшує швидкість передачі. Самодостатність означає, що токен містить всю необхідну інформацію про користувача, що дозволяє не робити запити до бази більше одного разу. JVT складається з трьох частин:

- назва. Він містить дві частини: тип маркера (JVT) і використовуваний алгоритм хешування (наприклад, HMAC SHA256 або RSA). Заголовок кодується за допомогою Base64Url і представляє першу частину JVT;

- вантажопідйомність. Містить інформацію про користувача та додаткові метадані. Є зарезервовані, публічні та приватні поля. Корисне навантаження також кодується Base64Url і представляє другу частину JVT;

- підпис Для створення підпису закодований заголовок і корисне навантаження, а також секретне слово беруться та підписуються за допомогою алгоритму, зазначеного в заголовку. Підпис використовується для того, щоб переконатися, що відправник JVT є тим, за кого себе видає, і що повідомлення не було змінено на шляху до сервера.

Під час використання JVT для автентифікації, коли користувач успішно входить із своїми обліковими даними, сервер повертає веб-токен JSON, який клієнт повинен зберігати локально. Цей підхід замінює традиційний, коли сеанс створюється на сервері, а його ідентифікатор повертається клієнту. Коли користувачеві потрібно отримати доступ до захищеної адреси або ресурсу, він надсилає JVT, наприклад, у HTTP-заголовку «Авторизація». JVT — це механізм аутентифікації без стану, тому що стан користувача не потрібно зберігати на сервері, токен містить всю необхідну інформацію. JVT має передаватися через безпечне з’єднання за допомогою, наприклад, HTTPS.

Це пов’язано з тим, що криптографічний алгоритм лише забезпечує перевірку того, що токен дійсно був згенерований сервером у тому вигляді, в якому він був отриманий.

Коли ви використовуєте відкритий протокол, третя сторона може легко побачити вміст повідомлення. За цією схемою пристрої надсилають свій ідентифікатор і пароль на спеціальну адресу для підключення до платформи. Сервер перевіряє в адміністративній базі даних, чи знає він про такий пристрій і чи повинен він з ним працювати. У разі позитивного рішення сервер надсилає на пристрій JVT, який записує додаткову інформацію про пристрій та його права доступу. Також доцільно включити тип пристрою в його корисне навантаження при формуванні JVT. Потім, щоб надіслати дані на сервер, клієнтське програмне забезпечення пристрою додає веб-токен JSON до HTTP-заголовка кожного запиту.

Коли він отримує запит, що містить маркер, сервер лише перевіряє його цифровий підпис за допомогою відповідного алгоритму. Маркер містить усю необхідну інформацію: ідентифікатор пристрою, який вказує, де були отримані певні дані, і тип пристрою, за допомогою якого зовнішній сервер може надсилати дані - надішліть повідомлення у відповідну тему в черзі повідомлень. Тому для обробки повідомлення не потрібно робити додаткові запити до бази даних; це значно збільшує пропускну здатність системи.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Схема розміщення системи

Щоб розгорнути платформу, ви можете вибрати один із хмарних хостів, наприклад Amazon Web Services або Google Cloud Platform. Але динамічне масштабування сервісів вимагає спеціальних рішень, які забезпечують роботу системи в цілому. Перш за все, коли потрібно розгорнути більше одного фронтенд-сервера, виникає необхідність розгорнути балансувальник навантаження, тобто інструмент для розподілу навантаження на кілька серверів.

Балансування навантаження

Для балансування навантаження ми будемо використовувати HAProki. HAProki — це серверне програмне забезпечення для забезпечення високої доступності та балансування навантаження для програм TCP і HTTP шляхом розподілу вхідних запитів між декількома серверами [29]. Програма написана мовою S.

HAProki є відкритим вихідним кодом і розповсюджується за загальною публічною ліцензією GNU (GNU GPL v2). Особливості HAProki:

- періодична перевірка доступності внутрішніх серверів, на які пересилаються запити користувачів. Дуже корисно, оскільки ця функція виключає сервер із пулу, до якого розповсюджуються запити, як тільки він перестає бути доступним з будь-якої причини;

- Підтримка HTTPS (TLS) і HTTP/2. Як уже було сказано, захищений (TLS) варіант HTTP/2 є основним протоколом зв’язку клієнтського програмного забезпечення з зовнішніми серверами;

- підтримка IPv6, стиснення HTTP (deflate, gzip, libslz), постійне з'єднання HTTP;

- Підтримка WebSocket. Що стосується продуктивності, то з тестів відомо, що HAProki на серверах, оснащених процесором Xeon E5 (2014 року випуску), здатний забезпечити потік даних до 40-60 Гбіт/с. Він розроблений якомога ефективнішим і використовує мінімальний час ЦП, тому продуктивність мережевої карти наразі є обмежуючим фактором, а не ЦП чи програмний балансувальник навантаження.

Коли ви досягаєте певного ліміту продуктивності HAProki, ви можете реалізувати ще більш ефективне рішення, яке масштабується майже необмежено – балансування на рівні DNS. HAProki надає можливість відновлювати сесії (закріплення сесії), що дозволяє надсилати запити конкретного клієнта на відповідний сервер, який обслуговує свою сесію в сесії. У балансуванні DNS такої функції немає, але платформі вона не потрібна: уся архітектура, включаючи автентифікацію, побудована так, щоб не зберігати стан між запитами.

Таким чином, ви можете реалізувати найпростіший алгоритм Round Robin DNS, який пересилає запити один за одним на сервери зі списку.

Пошук сервісів

Якщо конкретне клієнтське програмне забезпечення викликає певний сервер, йому, очевидно, потрібно знати його IP-адресу та порт для цього. У традиційних програмах, які працюють на невеликій кількості власних серверів компанії, мережеві розташування служб досить статичні. Їх можна прочитати з деякого файлу конфігурації, який час від часу оновлюється. У сучасних хмарних системах з мікросервісною архітектурою дізнатися IP і порт потрібного сервісу набагато складніше.

Мережеві адреси призначаються серверам динамічно. Крім того, набір послуг постійно змінюється через автомасштабування або збої. Тому необхідний більш складний механізм пошуку послуг. Відмінне рішення - Consul. Consul – розподілена система конфігурації та пошуку сервісу, яка має такі можливості [32]:

- пошук служб через DNS або HTTP. Деякі клієнти Consul надають (реєструють) послугу, а інші можуть подати запит і знайти її;

- Огляд здоров'я. Така інформація може використовуватися для моніторингу працездатності кластера або для перенаправлення трафіку з сервера, що має проблеми;

- сховище ключ/значення. Його можна використовувати для налаштування або налаштування служб;

- підтримка багатьох ЦОД. На кожному вузлі, який надає консульські послуги, працює агент. Він відповідає за тестування продуктивності. Агенти спілкуються з одним або кількома серверами Consul.

Дані зберігаються та тиражуються на серверах. З реплік вибирають головну. Consul може працювати з одним сервером, але щоб уникнути сценаріїв відмови сервера, які призводять до втрати даних, рекомендується використовувати від 3 до 5. Кожен центр обробки даних повинен мати окремий кластер Consul.

Будь-який компонент інфраструктури, якому потрібно знайти службу, може зробити запит до будь-якого сервера або агента. Агенти автоматично перенаправляють запити на сервери. Consul вимагає кворуму реплік для генерації відповідей на запит, щоб забезпечити надійну узгодженість. При наявності мережевого поділу Consul жертвує доступністю, тобто, з точки зору теореми CAP, це система CP.

Порівняно з іншими подібними рішеннями, головною перевагою Consul є те, як він виконує перевірку працездатності сервера. Більшість альтернативних систем надсилають мережеві запити службам, щоб визначити їхній статус. Такі перевірки надають набагато менше інформації, ніж консульські агенти, які працюють на кожному сервері, і мають можливість збирати набагато детальнішу інформацію про систему.

Традиційні реалізації перевірки працездатності також вимагають обсягу роботи, який лінійно залежить від кількості вузлів у кластері, тому погано масштабується.

Клієнти Consul використовують протокол gossip для передачі даних про стан сервера, що дозволяє масштабувати кластери будь-якого розміру, не зосереджуючи основне навантаження на певній групі серверів.

Розгортання і масштабування

Рішення щодо масштабування можна приймати за допомогою спеціальних інструментів хмарного хостингу, таких як Amazon Auto Scaling [30]. Amazon Auto Scaling може запускати нові сервери, коли перевищено певний поріг навантаження ЦП або використання оперативної пам'яті. Крім того, цей інструмент може зупинити частину сервера, якщо кластер споживає мало ресурсів. Після запуску сервера (початкового, в результаті масштабування або при відновленні після збою) необхідно встановити на цей сервер необхідне програмне забезпечення і запустити відповідну службу, тобто якийсь виконуваний код.

Існує два основних підходи до налаштування:

- Push розгортання. Після запуску сервера на нього передається необхідний виконуваний код, наприклад, за допомогою rsync (Unik);

- Витягнути розгортання. Сервер запускається з певною конфігурацією, яка визначає, звідки можна завантажити виконуваний код. Сервер запитує його з певного центрального сховища, збирає та запускає.

Push-розгортання більше підходить для невеликих кластерів фіксованого розміру. Динамічне масштабування вимагає підходу залучення, розміщення виконуваного коду служби в центральному репозиторії, щоб, коли сервер автоматично запускається, він сам розгортав службу. Щоб запустити службу, ви також повинні встановити необхідні залежності, такі як JRE для програм Java. Залежностей може бути багато, тому цей аспект розгортання також має бути

автоматизованим. Зручним і функціональним рішенням для розгортання автоматизації є Docker.

Docker - це утиліта з відкритим кодом, яка автоматизує розгортання додатків у програмних контейнерах, забезпечуючи додатковий рівень абстракції та автоматизації віртуалізації на рівні ОС у Linux [31].

Docker реалізує API високого рівня для легких контейнерів, які ізольовано запускають процеси. Створений для використання можливостей, наданих ядром Linux (інструменти контрольних груп і просторів імен), контейнер Docker, на відміну від віртуальної машини, не вимагає (і не містить) окремої операційної системи. Замість цього він використовує ізоляцію ресурсів (ЦП, пам’ять, введення/виведення, мережу тощо) і окремі простори імен для ізоляції програми в операційній системі. Це дозволяє створити розділ операційної системи для програм, який має власний простір ідентифікатора процесу, а також структуру файлової системи та мережеві інтерфейси. Різні контейнери мають одне ядро, але кожен з них може бути обмежений у використанні ресурсів.

Використання Docker для створення контейнерів і керування ними спрощує створення розподілених систем, дозволяючи програмам працювати автономно на окремих серверах або в кластері. Екосистема Docker, окрім основного модуля Docker Engine, включає такі компоненти:

- Docker Compose – дозволяє легко описати та запустити конфігурацію багатоконтейнерної програми з усіма її залежностями;

- Docker Swarm – інструмент управління кластером, який дозволяє логічно об'єднати кілька движків Docker в один;

- Docker Registries – утиліта для зберігання та розповсюдження образів Docker. Корисно для організації відкатів - сервери завантажуватимуть зображення з цього реєстру. Покажемо схему розміщення системи.

Consul надає послуги моніторингу та пошуку зовнішніх серверів, а Zookeeper потрібен для координації внутрішніх вузлів Kafka. Робота MongoDB

забезпечується одним головним сервером і одним або двома підлеглими серверами. Розташування Спарка і Кассандри також є важливим аспектом. Оскільки більшість даних для виконання обчислень береться з оперативної бази даних, рекомендується запускати Spark Worker і вузол Cassandra на одному фізичному сервері. Spark, а також драйвер для його підключення до Cassandra (Spark Cassandra Connector) мають широкі можливості для організації обчислень таким чином, щоб підвищити локалізацію даних, тобто максимально зменшити рух даних мережею.

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено для побудови системи платформи Інтернету речей.

Метою розробки є дослідження та програмна реалізація розподіленої сервісно-орієнтованої платформи Інтеренету речей яка забезпечить високу продуктивність і захист інформації.

Об'єктом дослідження є процес побудови платформи Інтернету речей.

Предметом дослідження є методи реалізації серверних систем платформ Інтернету речей.

Методи дослідження базуються на методах теорії кодування, методах математичної статистики,методах захисту інформації, методах розробки програмного забезпечення та методах побудови та обслуговування мережних пристроїв.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– ефективна реалізація компонентів системи дає змогу оптимізувати витрати на апаратне забезпечення платформи;

– платформа також гарантує значний рівень відмовостійкості, що знижує ймовірність втрати операційних даних і забезпечує високий рівень її доступності.;

– досліджені та реалізовані аспекти безпеки надають високий рівень захищеності даних при передачі й убезпечують систему від різноманітних інформаційних атак.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація системи захисту в інформаційній системі.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	120
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	120000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \text{ Size}^B, \quad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \quad (7.2)$$

де: Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \tag{7.3}$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$
$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \tag{7.4}$$

де: C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 47 = 79 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	79	Ф 7.1-7.4
Впровадження	13	Д13
Всього	120	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{pq} - H_{ев}}, \quad (7.5)$$

де: F_{pq} – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{120 \cdot 1}{60 - 5} = 2,2 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор–маршрутизатор	30	2	60	1
Кабельні господарства ЛВС на 1 м. п.	2,5	200	500	8,33
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	39,49

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{40 \cdot 3}{1,2} = 100 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 100/(60 \cdot 8) = 0{,}2 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2022, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	15974	47922
Продакт-менеджер	0,25	12000	9000
Інженер-програміст	2,2	15000	99000
Інженер-електронщик	0,2	12000	7200
Інженер-системотехнік	0,25	12000	9000
Адміністратор мережі	0,5	12000	18000
Системний програміст	0,25	12000	9000
Дизайнер WEB	0,25	12000	9000
Інженер-верстальник	0,25	12000	9000
Бухгалтер-економіст	0,5	13000	19500
Всього за період розробки	$R_{сп}$ = 5,65	-	$Ф_{роб}$ = 236622

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рq}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{236622}{5,65 \cdot 60} = 698 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пектораль» (м. Кіровоград) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 37 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно $8 м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Brain за 26.10.23 – джерело http: //brain.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	Intel Pentium G6405 (BX80701G6405) 1200, 2 ядра, 4 потоки, 4.1 GHz, TDP - 58 Вт, 14nm box	
Системна плата	GIGABYTE H510M K V2 DDR4, LAN - 1 Гбіт/с, HDMI, 1 x M.2 2280, 2 x Sata 6.0 Gb/s, Micro-ATX	
Відеокарта	Intel UHD Graphics 610	
Жорсткий диск	SSD M.2 2280 512GB LEVEN (JP600PCIE512GB) Серія - JP600, 512 GB, 3D TLC NAND, M.2, PCI Express 3.0 x4	
Оперативна пам’ять	DDR4 8GB 3200 MHz Fury Beast Black Kingston Fury (ex.HyperX) (KF432C16BB/8) DDR4, 8 ГБ, В наборі - 1	
Блок живлення	Gamemax 500W (GM-500B) ATX 12V v2.3, 500 Вт, 20+4 pin, CPU - 4+4pin, GPU - 1x6 pin, SATA - 3, Peripheral - 2, +12V1 - 20A, 1x120 мм, 150 x 140 x 86 мм	
Корпус	Vinga CS210B, Miditower, ATX, Micro - ATX, Mini - ITX, Слотів розширення - 7, Додатково - Front 2x12 см, Back 1x12 см, Side 2x12 см	

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	220
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	15	10947	16420,5	180625,5
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	-	-	-	0
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	199177

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	199177	-	-
Всього по групі	199177	50	99588,5
Група 5, 6			
4. Вимірювальні пристрої	5190	25	1297,5
5. Транспортні засоби	0	20	0,0
6. Господарський інвентар	28000	25	7000
Всього по групі	33190	-	8297,5
7. Нематеріальні активи	120000	10	12000
Разом	$К_р$ = 1760367		$А_р$ = 190286

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 698 \cdot 120/120 = 698 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 698 \cdot 10 \cdot 0{,}01 = 70 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(698+70) = 288 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 698 \cdot 15 \cdot 0{,}01 = 105 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картриджів, тонеру, грн.;

N_e – кількість екземплярів програм, шт.

Згідно виданих норм приймаємо 1/6 пачки паперу на місяць розробки (n_p=1/6). Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$ = 210 грн., визначаємо вартість паперу за період розробки $N_м$ = 3 міс:

$$З_{М1} = Ц_п \cdot n_р \cdot N_м. \quad (7.16)$$

$$З_{М1} = 210 \cdot 1/6 \cdot 3 = 105 \text{ грн.}$$

Згідно виданих норм до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків в кількості 50 примірників:

$$З_{М2} = \sum Ц_д , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 28,8 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 28,8 грн./шт.

$$З_{М2} = 28{,}8 \cdot 50 = 1440 \text{ грн.}$$

Згідно виданих норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (105 + 1440 + 1702)/120 = 27 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 698 \cdot 15 \cdot 0{,}01 = 105 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм (N_e = 120 прим.):

$$A_м = \frac{A_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де: $A_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = 190286 \cdot 3/(120 \cdot 12) = 396 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$C_п = З_о + З_д + C_{оц} + Г_{осп} + З_м + О_п + A_м. \quad (7.21)$$

$$C_п = 698+70+288+105+27+105+396 = 1689 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1. Основна зарплата виконавців	$З_о$	698
2. Додаткова зарплата виконавців	$З_д$	70
3. Відрахування на соціальні потреби	$C_{оц}$	288
4. Загальногосподарські витрати	$Г_{осп}$	105
5. Витрати на матеріали	$З_М$	27
6. Освоєння нових операційних систем, мов програмування	$О_п$	105
7. Амортизація основних фондів	$A_м$	396
8. Повна собівартість програмного забезпечення	$C_п$	1689
9. Плановий прибуток	$П_р$	929
10.Ціна підприємства $Ц_п = C_п + П_р$	$Ц_п$	2618
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	523,6
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	3141,6

Визначимо плановий прибуток за рівнем рентабельності (P_n) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 55%.

$$П_р = 0{,}01 \cdot P_n \cdot C_n, \quad (7.22)$$

де: P_n – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 55 \cdot 1689 = 929 \text{ грн.}$$

7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

Об'єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	3142
Всього капітальних витрат	–	3142

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування системи	$З_р$	40260	20130
2. Витрати на електроенергію	$З_{ел}$	621	311
3. Витрати на амортизацію	$З_{ам}$	0	786
Всього витрат за рік	$І$	40881	21227

Витрати на обслуговування роботи системи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість годин на обслуговування системи зменшилось з 300 год до 150 год на рік.

$$З_{р\,баз} = 300 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 40260 \text{ грн},$$

$$З_{р\,нов} = 150 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 20130 \text{ грн}.$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	3142	–	785,5
Всього відрахувань	-	–	3142	–	785,5

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($T_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot T_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}545 \cdot 300 \cdot 3{,}8 = 621 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}545 \cdot 150 \cdot 3{,}8 = 311 \text{ грн.}$$

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (2618 - 1689) \cdot 120 - (0{,}05 \cdot 1408000 + 0{,}4 \cdot 199177 +$$
$$+0{,}25 \cdot 33190 + 0{,}1 \cdot 120000) \cdot 3/12 = 68888 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1760367}{(2618 - 1689) \cdot 120 \cdot 12 / 3} = 3{,}95 \text{ роки}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (40881\text{-}21227)\text{-}0{,}25 \cdot 3142 = 18869 \text{ грн.}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	120
2. Повна собівартість розробленої програми	Грн.	1689
3. Ціна розробленої програми	Грн.	2618
4. Плановий прибуток від реалізації розробленої програми	Грн.	929
5. Рентабельність програмної продукції	%	55
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1760367
7. Загальний прибуток від реалізації програмної продукції	Грн.	111480
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	68888
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Роки	3,95
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	3142
11.Величина економічного ефекту у користувача програмної продукції	Грн.	18869
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,16

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \tag{7.28}$$

$$T_{сп} = \frac{3142}{40881 - 21227} = 0{,}16 \ року .$$

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Вступ

Протягом усієї історії людство приділяє прискіпливу увагу безпеці життя. Охорона праці є складовою частиною безпеки життя.

Законом України "Про охорону праці" регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та

«Правила охорони праці під час експлуатації електронно-обчислювальних машин» НПАОП 0.00-1.28-10,

Умови праці програміста включають наступні фактори:

- параметри повітряного середовища в приміщенні;
- вентиляція приміщення;
- освітлення приміщення;
- параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення впливу комп'ютера на організм програміста визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.1 Шкідливі і небезпечні фактори при роботі з комп'ютером

Програміст працює з електронно-обчислювальною машиною (ЕОМ) та іншим обладнанням, яке є джерелом небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. Так як програміст постійно перебуває в приміщенні, тому для комфортних умов праці в цьому приміщенні необхідно створити належний мікроклімат.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.
- ризик виникнення пожежі;
- негативний вплив на органи зору людини;
- ризики ураження електричним струмом;
- недостатня, або надмірна освітленість робочого місця;
- монотонність праці;
- електромагнітні електромагнітні (у т.ч. високочастотні) випромінювання (коливання);

- несприятливі мікрокліматичні умови;

- нервово-емоційна напруженість праці;

- інтелектуальні навантаження;

- невідповідність ергономічних показників робочого місця діючим вимогам;

- шуми;

- статичні навантаження на кістково-м’язовий апарат;

8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 - Розміри приміщення

Найменування	Значення, м
Ширина	2,4
Довжина	3
Висота	2,8

Таблиця 8.2 - Площа та обсяг приміщення, на одного працюючого

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	7,2
Обсяг, V	$м^3$	не менше 20.0	20,1

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працює 1 людина. За даними, які наведено у табл. 8.1 та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста відповідають нормативним вимогам (Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, яка виконується в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря у приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

Таблиця 8.3 - Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Вологість, %	Швидкість повітря, м/с	Температуа, °С	Вологість %	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23	41-59	0,1
Тепла	23-25	50-70	0,1	23,5-24	53-70	0,13

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер *HP LaserJet 1018*, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5 – 28 – 2006 р. можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення

коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.3 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;
- мікроклімат відповідає нормативному значенню;
- акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору розтіканню електричного струму на землю).

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.4 Розрахункова частина

Початкові данні для розрахунку штучного захисного заземлення:

Тип заземлення: робоче заземлення нульової точки трансформатора. Напруга — 220/380 В. Розташування заземлюючих електродів — по контуру.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача методом коефіцієнта використання заземлювачів.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта — чорнозем, нижнього шару грунта — глина. Умовна товщина верхнього шару грунта: *H=0,8 м*. Для захисного заземлення: застосовуються ветикальні електроди — пруток довжиною *L=2 м*. Відстань між вертикальними заземлювачами (електродами) *A*=2 *м*. Діаметр ветикальньного електрода (прутка) *D=60 мм*, Тип горизонтального заземлювача: металева полоса. Розміри перетину з'єднуючої полоси: *60 х 6 мм*. (*b=60 мм*.). Опір заземлювача, який нормується: $R_{ЗН}$ = *4 Ом*. Глибина закладення горизонтального контура заземлення *t= 0,6 м*.

Розрахунок захисного заземлення можна автоматизувати за допомогою програми, сирцевий код якої опублікован на стр.13-16 Метододичних вказівок до виконання розрахунків з викор. персон. ЕОМ IBM сумісного типу / Охорона праці. Ч. 1. Захисне заземлення / Кіровоград. ін-т с.-г. Машинобуд.; URL : http://dspace.kntu.kr.ua/jspui/handle/123456789/4358, або будь якої відповідної іншої.

Розрахунок

Відстань від центра вертикального заземлювача до поверхні землі:

T=t+L/2=0,6+2/2=1,6 м.

Еквівалентний питомий опір грунта:

$\rho_{екв} = \psi \rho_1 \rho_2 L / [\rho_1 \psi(L-H+t) + \rho_2 \psi(H-t)] = 93,75$ *Ом*

де $\psi = 1,2$ - табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому грунті (джерело: http://ftemk.mpei.ac.ru/bgd/_private/Grunt_/t_goda.htm);

$\rho_1 = 50$ *Ом*м.* - табличне значення питомого опору верхнього шару грунта (джерело: https://el-line.ru/udelnoe_soprotivlenie_grunta.shtml);

$\rho_2 = 100$ *Ом*м.* - табличне значення питомого опору нижнього шару грунта (джерело: https://zandz.com/ru/udelnoe_soprotivlenie_grunta).

Опір розтіканню електричного струму одного електрода вертикального заземлювача:

$R_O = [\rho_1/(2\pi*L)]*[ln(2L/D)+0,5ln(4T+L)/(4T-L)] = 40,6$ *Ом.*

Опір розтіканню електричного струму заземлювача, який нормується, $R=1$ якщо $\rho_{екв} < 100$ *(у нас* $\rho_{екв} = 93,75 < 100$*),* та $R = R_{ЗН} * \rho_{екв}/100$, якщо $\rho_{екв} > 100$.

Опір розтіканню електричного струму горизонтального заземлювача (полоси):

$$R_П = 0,366(\rho_{екв}\, \psi / L_П \cdot \eta_П) \lg(2 / L_П * L_П / bt) = 35,7 \text{ Ом.}$$

де $\eta_П = 0,3$ - табличне значення коефіцієнта використання горизонтального

заземлювача, залежить від розташування (в ряд або по контуру) та співвідношення A/L;

довжина горизонтального заземлювача (полоси) при розташуванні заземлювачів по контуру:

$L_П = A*n = 2*16 = 32$ *м.;*

де n – ітераційна кількість вертикальних заземлювачів.

Загальний опір штучного заземлюючого пристрою розтіканню електричного струму на землю:

$$R_B = R_П R / (R_П - R) = 4,5 \text{ Ом.}$$

Враховуючи, що значення ρ_1 *та* ρ_2 по факту можуть бути меншими, розрахункове значення R_B цілком прийнятне.

Кількість вертикальних заземлювачів:

$$n = R_O / R_B * \eta_B = 16шт.$$

де $\eta_B = 0,6$ - табличне значення коефіцієнта використання вертикального

заземлювача, залежить від розташування (в ряд або по контуру) та співвідношення A/L.

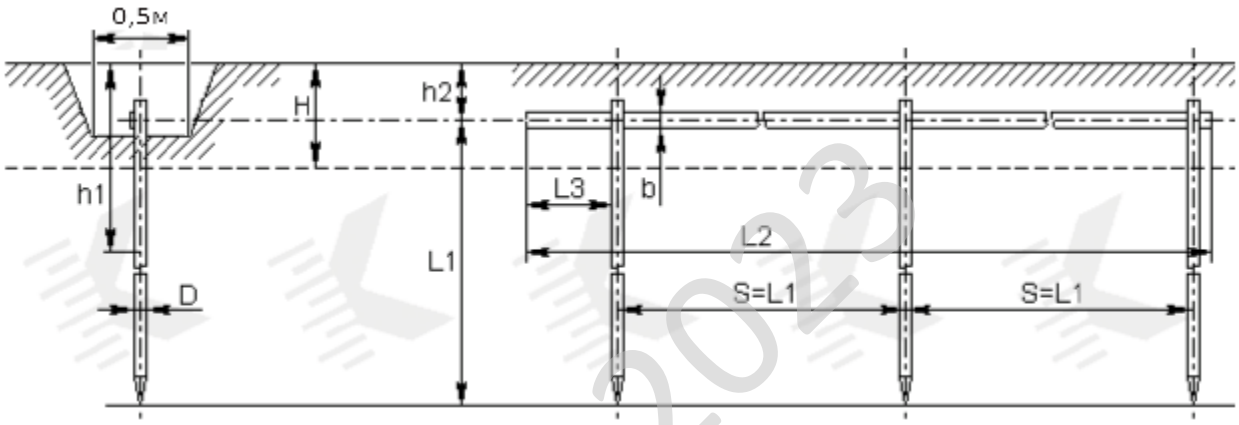

Рисунок 8.1 — Штучний заземлювач

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для дослідження та побудови платформ для Інтернету речей.

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

Зокрема, було виділено основні вимоги, що висуваються до подібних систем; вивчено основні підходи до вирішення задач обробки великих за обсягами даних; досліджено існуючі платформи та виокремлено шляхи їх покращення.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження методів .

Розроблена платформа задовольняє поставленим функціональним і нефункціональним вимогам та містить реалізацію таких підсистем: приймання, зберігання, обробки операційних даних; автентифікації та безпеки; роботи з адміністративними даними; моніторингу.

Це дозволяє застосування її такими виділеними групами користувачів: пристроями, адміністраторами, аналітиками і адміністраторами безпеки. Було запропоновано та реалізовано таку розподілену сервісно-орієнтовану (мікросервісну) архітектуру платформи, яка забезпечує високу продуктивність і майже нескінченну масштабованість. Це дозволяє нарощувати потужність системи для обслуговування мереж пристроїв будь-яких розмірів і для збільшення можливостей підсистеми аналітики даних.

Ефективна реалізація компонентів системи дає змогу оптимізувати витрати на апаратне забезпечення платформи.

Платформа також гарантує значний рівень відмовостійкості, що знижує ймовірність втрати операційних даних і забезпечує високий рівень її доступності.

Досліджені та реалізовані аспекти безпеки надають достатній рівень захищеності даних при передачі й убезпечують систему від різноманітних інформаційних атак.

У ході розробки платформи були з належним обґрунтуванням відібрані та застосовані сучасні системи, протоколи, формати даних, бібліотеки та бази даних.

Використанні сторонні рішення мають відкритий вихідний код, що спрощує розширення системи, має економічні та інші переваги.

Було детально описано різноманітні аспекти обраних архітектурних рішень: від обґрунтування високорівневого поділу на функціональні модулі до особливостей ефективної реалізації вводу/виводу та впливу обраної схеми автентифікації на масштабованість архітектури платформи.

На відміну від існуючих платформ, розроблене рішення є добре розширюваним у тому сенсі, що його можливо легко адаптувати під аналітику даних, які надходять від різноманітних пристроїв і сенсорів, задаючи ланцюжки обробки даних. Протоколи та формати даних було обрано не лише з огляду на їх ефективність, а ще й враховуючи зручність і простоту розширення платформи.

Іншими ключовими особливостями платформи є моніторинг і функціональність надійної передачі службової інформації між пристроями та адміністраторами. Ці можливості дозволяють ефективно керувати мережею пристроїв, а також гнучко налаштовувати інструменти попередження про можливі неполадки та оперативно на них реагувати.

Запропоновані та застосовані в роботі архітектурні та технічні рішення є достатньо універсальними, що дозволяє використовувати їх для побудови різноманітних систем обробки великих даних та інших платформ для Інтернету

речей, спеціалізованих для більш вузьких областей. Роботу платформи було показано за допомогою демонстраційного прикладу, в якому моделюється робота системи з мережею метеорологічних станцій.

Також до платформи було підключено мікрокомп'ютер Raspberry Pi 3 та продемонстровано обробку даних із його сенсора. Ці приклади охоплюють більшість можливостей системи та показують її застосування всіма групами користувачів.

Описано реалізацію на боці клієнта процесів реєстрації та автентифікації пристрою, а також організацію відправки даних до сервера. 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 Java та Python. Дані мови програмування дозволяю найбільш ефективно обробляти дані. 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.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

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.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 19869 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,16 років.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Доповідь компанії Gartner. – Режим доступу: http://www.gartner.com/newsroom/id/3165317. – Дата доступу: 22.09.2023.

2. Доповідь компанії Cisco. – Режим доступу: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoE_Economy.pdf. – Дата доступу: 23.05.2023.

3. Nathan Marz. Big Data: Principles and best practices of scalable realtime data systems / Nathan Marz, James Warren. – Manning Publications, 2015. – 328 p.

4. Arvind Sathi. Big Data Analytics: Disruptive Technologies for Changing the Game / Arvind Sathi. – Mc Press, 2012. – 96 p.

5. Vijay K. Garg. Elements of Distributed Computing / Vijay K. Garg. – Wiley-IEEE Press, 2002. – 448 p.

6. Gerard Tel. Introduction to Distributed Algorithms / Gerard Tel. – Cambridge University Press, 2000. – 612 p.

7. M. Tamer Özsu. Principles of Distributed Database Systems / M. Tamer Özsu, Patrick Valduriez. – Springer, 2011. – 846 p.

8. Refactoring: Improving the Design of Existing Code / [Martin Fowler, Kent Beck, John Brant et al.]. – Addison-Wesley Professional, 1999. – 464 p.

9. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship / Robert C. Martin. – Prentice Hall, 2008. – 464 p.

10. Design Patterns: Elements of Reusable Object-Oriented Software / [Erich Gamma, Richard Helm, Ralph Johnson et al.]. – Addison-Wesley Professional, 1994. – 395 p.

11. Martin Fowler. Patterns of Enterprise Application Architecture / Martin Fowler. – Addison-Wesley Professional, 2002. – 560 p.

12. Len Bass. Software Architecture in Practice / Len Bass, Paul Clements, Rick Kazman. – Addison-Wesley Professional, 2012. – 640 p.

13. Luke Hohmann. Beyond Software Architecture: Creating and Sustaining

Winning Solutions / Luke Hohmann. – Addison-Wesley Professional, 2003. –352 p.

14. Інформація про платформу AWS IoT. – Режим доступу: https://aws.amazon.com/iot/how-it-works/. – Дата доступу: 24.05.2016.

15. Інформація про платформу IBM IoT Platform. – Режим доступу: http://www.ibm.com/internet-of-things/iot-platform.html. – Дата доступу: 25.05.2016.

16. Інформація про платформу Oracle IoT. – Режим доступу: https://cloud.oracle.com/iot. – Дата доступу: 25.05.2016.

17. Специфікація HTTP/2 (Internet Engineering Task Force. Request for Comments 7540). – Режим доступу: https://tools.ietf.org/html/rfc7540. –Дата доступу: 26.05.2023.

18. Специфікація WebSocket (Internet Engineering Task Force. Request for Comments 6455). – Режим доступу: https://tools.ietf.org/html/rfc6455. –Дата доступу: 27.05.2023.

19. Sam Newman. Building Microservices: Designing Fine-Grained Systems / Sam Newman. – O'Reilly Media, 2015. – 280 p.

20. Офіційна документація Apache Kafka. – Режим доступу: http://kafka.apache.org/documentation.html. – Дата доступу: 29.05.2023.

21. Офіційна документація Apache Spark. – Режим доступу: http://spark.apache.org/docs/latest/. – Дата доступу: 31.05.2023.

22. Інформація про Sort Benchmark і його результати. – Режим доступу: http://sortbenchmark.org/. – Дата доступу: 31.05.2023.

23. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing / [Matei Zaharia, Mosharaf Chowdhury, Tathagata Das et al.]. – Режим доступу: https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf. – Дата доступу: 31.05.2023.

24.Офіційна документація Apache Cassandra. – Режим доступу: https://wiki.apache.org/cassandra/. – Дата доступу: 02.06.2023.

25. Dynamo: Amazon's Highly Available Key-value Store / [Giuseppe DeCandia, Deniz Hastorun, Madan Jampani et al.]. – Режим доступу:

http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf. – Дата доступу: 02.06.2023.

26. Niels Provos. A Future-Adaptable Password Scheme / Niels Provos, David Mazières // Proceedings of 1999 USENIX Annual Technical Conference. –1999. – P. 81-92.

27. Офіційна документація MySQL. – Режим доступу: http://dev.mysql.com/doc/refman/5.7/en/. – Дата доступу: 02.06.2023.

28 .Специфікація JSON Web Token (Internet Engineering Task Force. Request for Comments 7519). – Режим доступу: https://tools.ietf.org/html/rfc7519. – Дата доступу: 03.06.2023.

29. Офіційна документація HAProxy. – Режим доступу: http://www.haproxy.org/. – Дата доступу: 04.06.2023.

30.Офіційна документація Amazon Web Services. – Режим доступу: https://aws.amazon.com/documentation/. – Дата доступу: 04.06.2023.

31.Офіційна документація Docker. – Режим доступу: https://www.docker.com/. – Дата доступу: 04.06.2023.

32.Офіційна документація Consul. – Режим доступу: https://www.consul.io/. – Дата доступу: 04.06.2023

33. IoT Based Home Security System Using Raspberry Pi with Email and Voice Alert. Reena Rani, S. Lavanya,. B.Poojitha Published Computer Science, 2018 pp 21-26

34. Home Security System using IOT and AWS Cloud Services.Mahendra Mehra; Vedant Sahai; Pratik Chowdhury; Elvis Dsouza. International Conference on Advances in Computing, Communication and Control (ICAC3), 2019 pp 159-163

35. Anitha A, “Home security system using internet of things”, IOP Conf. Series: Materials Science and Engineering 263, 2017 pp 1-11

36. OneM2M Architecture Based Secure MQTT Binding in Mbed OS. Ahsan Muhammad, Bilal Afzal, Bilal Imran, Asim Tanwir, Ali Hammad Akbar, Ghalib A. Shah. Computer Science. IEEE European Symposium on Security and Privacy, 2019

pp 48-56.

37. Heapster: Compute Resource Usage Analysis and Monitoring of Container Clusters [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/kubernetes/heapster

38. Performance Testing Guidance for Web Applications by Microsoft Corporation Microsoft Corporation., 2007. – 221 с. – (Microsoft Corporation).

39. The httperf HTTP load generator [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/httperf/httperf.

40. Docker [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Docker.

41. Стек MEAN [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MEAN_(веброзробка)

42. Веб-технології для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web

43. jQuery [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JQuery

44. Linux [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Linux

45. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

46. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

47. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/

48. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

49. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

50. М. Кантелон , М. Хартер, Т. Головайчук, Н. Райлих // “Node.js в деї.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення системи захисту в інформаційній системі.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 32-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою роботи є дослідження та програмна реалізація розподіленої сервісно-орієнтованої платформи Інтеренету речей яка забезпечить високу продуктивність і захист інформації.

4 Джерела розробки

Джерелом цієї магістерської роботи є відносна 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:

– Розробку додатку;

– систему підключення та тестування баз даних ;

– 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 архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

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 XP/Vista/7/8/10/11.

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 Java та Scala

Бази даних SQL та NOSQL

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 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці та техніки безпеки в магістерській роботі повинні бути розглянуто заходи покращення умов праці та розрахунок захисного заземлення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 121 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі бакалаврської дипломної роботи. Постановка задачі на виконання бакалаврської дипломної роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень бакалаврської дипломної роботи.

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

11.1 Подання бакалаврської дипломної роботи на попередній захист 10.12.2023 р.

1.2 Подання магістерської роботи на захист 20.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Пархоменко Ю.М.

Дослідження та програмна реалізація системи захисту в інформаційній системі

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 16

Літера: РП

Кропивницький – 2021 року

Файл основного файлу програми

MessageQueueSender.java

```
package com.iot.mq;
import java.util.concurrent.CompletionStage;
public interface MessageQueueSender {
CompletionStage<Void> send(String topic, String key, String data);
}
```

KafkaSender.java

```
package com.iot.mq;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
import java.util.Properties;
import java.util.concurrent.CompletableFuture;
public class KafkaSender implements MessageQueueSender {
private final Producer<String, String> producer;
public KafkaSender() {
Properties properties = new Properties();
properties.put("bootstrap.servers", "localhost:9092");
properties.put("acks", "1");
properties.put("key.serializer", StringSerializer.class.getName());
properties.put("value.serializer", StringSerializer.class.getName());
producer = new KafkaProducer<>(properties);
}
public CompletableFuture<Void> send(String topic, String key, String data) {
ProducerRecord<String, String> record = new ProducerRecord<>(topic,
key, data);
CompletableFuture<Void> future = new CompletableFuture<>();
producer.send(record, (metadata, exception) -> {
if (exception != null) {
future.completeExceptionally(exception);
} else {
future.complete(null);
}
```

```
});
return future;
}
}
```

AuthenticationService.java

```
package com.iot.auth;
import java.util.concurrent.CompletionStage;
public interface AuthenticationService {
CompletionStage<Boolean> authenticate(String login, String password);
}
```

MongoAuthenticationService.java

```
package com.iot.auth;
import com.mongodb.async.client.MongoClient;
import com.mongodb.async.client.MongoClients;
import com.mongodb.async.client.MongoCollection;
import com.mongodb.async.client.MongoDatabase;
import org.bson.Document;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionStage;
import static com.mongodb.client.model.Filters.eq;
public class MongoAuthenticationService implements AuthenticationService {
public static final String DB_NAME = "iot";
public static final String COLLECTION_NAME = "device_info";
public static final String DEVICE_ID = "device_id";
public static final String PASSWORD = "password";
private final MongoCollection<Document> deviceInfoCollection;
public MongoAuthenticationService() {
MongoClient mongoClient = MongoClients.create();
MongoDatabase db = mongoClient.getDatabase(DB_NAME);
deviceInfoCollection = db.getCollection(COLLECTION_NAME);
}
@Override
public CompletionStage<Boolean> authenticate(String login, String password) {
CompletableFuture<Boolean> future = new CompletableFuture<>();
deviceInfoCollection.find(eq(DEVICE_ID, login)).first((result,
exception) -> {
```

```
if (exception != null) {
future.completeExceptionally(exception);
return;
}
if (result == null) {
future.complete(false);
return;
}
Object pwd = result.get(PASSWORD);
if (pwd instanceof String && pwd.equals(password))
future.complete(true);
115
else
future.complete(false);
});
return future;
}
}
```

JwtTokenService.java

```
package com.iot.token;
import com.auth0.jwt.JWTSigner;
import com.auth0.jwt.JWTVerifier;
import java.util.Map;
import java.util.regex.Pattern;
public class JwtTokenService {
private static final String SECRET = "feRtfVqoyDJWVVoP4X2m";
private static final Pattern AUTH_HEADER_PATTERN = Pattern.compile("^Bearer$",
Pattern.CASE_INSENSITIVE);
private final JWTVerifier verifier = new JWTVerifier(SECRET);
private final JWTSigner signer = new JWTSigner(SECRET);
public String createToken(Map<String, Object> claims) {
return signer.sign(claims);
}
public Map<String, Object> verifyToken(String authorizationHeader) throws
TokenParseException,
TokenVerificationException {
```

```
String token = extractToken(authorizationHeader);
try {
return verifier.verify(token);
} catch (Exception e) {
throw new TokenVerificationException(e);
}
}
private static String extractToken(String authorizationHeader) throws
TokenParseException {
if (authorizationHeader == null)
throw new TokenParseException("Authorization header value is
null");
String[] parts = authorizationHeader.split(" ");
if (parts.length != 2) {
throw new TokenParseException("Incorrect format: '" +
authorizationHeader +
"'. Format is Authorization: Bearer
[token]");
}
String scheme = parts[0];
String credentials = parts[1];
116
if (AUTH_HEADER_PATTERN.matcher(scheme).matches()) {
return credentials;
} else {
throw new TokenParseException("Incorrect scheme: " + scheme);
}
}
}
HttpUtils.java
package com.iot.http;
import io.undertow.server.HttpServerExchange;
import java.util.Deque;
import java.util.Map;
import static io.undertow.util.StatusCodes.BAD_REQUEST;
import static io.undertow.util.StatusCodes.INTERNAL_SERVER_ERROR;
```

```
public class HttpUtils {
public static Void sendRequestError(HttpServerExchange exchange) {
return sendError(exchange, BAD_REQUEST);
}
public static Void sendServerError(HttpServerExchange exchange) {
return sendError(exchange, INTERNAL_SERVER_ERROR);
}
public static Void sendError(HttpServerExchange exchange, int code) {
return sendError(exchange, "", code);
}
public static Void sendError(HttpServerExchange exchange, String body, int
code) {
exchange.setStatusCode(code);
exchange.getResponseSender().send(body);
return null;
}
public static String extractQueryParameter(String parameter, Map<String,
Deque<String>> queryParameters) {
Deque<String> values = queryParameters.get(parameter);
if (values == null || values.isEmpty())
return null;
return values.getFirst();
}
}
```


117

AuthenticationHttpHandler.java

```
package com.iot.http;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.iot.auth.AuthenticationService;
import com.iot.token.JwtTokenService;
import io.undertow.server.HttpHandler;
import io.undertow.server.HttpServerExchange;
import io.undertow.util.Headers;
import io.undertow.util.Methods;
import io.undertow.util.StatusCodes;
import java.util.Deque;
```

```
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.CompletionStage;
import static com.iot.http.HttpUtils.extractQueryParameter;
import static com.iot.http.HttpUtils.sendError;
import static com.iot.http.HttpUtils.sendServerError;
import static io.undertow.util.StatusCodes.UNAUTHORIZED;
public class AuthenticationHttpHandler implements HttpHandler {
public static final String DEVICE_ID = "deviceId";
public static final String LOGIN = "login";
public static final String PASSWORD = "password";
private final AuthenticationService authenticationService;
private final JwtTokenService tokenService;
private final ObjectMapper objectMapper;
public AuthenticationHttpHandler(AuthenticationService authenticationService,
JwtTokenService tokenService,
 ObjectMapper objectMapper) {
this.authenticationService = authenticationService;
this.tokenService = tokenService;
this.objectMapper = objectMapper;
}
@Override
public void handleRequest(HttpServerExchange exchange) throws Exception {
Map<String, Deque<String>> queryParameters =
exchange.getQueryParameters();
String login = extractQueryParameter(LOGIN, queryParameters);
String password = extractQueryParameter(PASSWORD, queryParameters);
if (login == null || password == null) {
sendError(exchange, "Login credentials are not present",
UNAUTHORIZED);
return;
}
exchange.dispatch();
CompletionStage<Boolean> authenticationFuture =
authenticationService.authenticate(login, password);
authenticationFuture
```

```
.thenAccept((authenticated) -> {
if (authenticated) {
createAndSendToken(login, exchange);
return;
118
}
sendError(exchange, "Authentication
credentials are invalid", UNAUTHORIZED);
})
.exceptionally(throwable ->
sendServerError(exchange));
}
private void createAndSendToken(String login, HttpServerExchange exchange) {
Map<String, Object> tokenClaims = new HashMap<>();
tokenClaims.put(DEVICE_ID, login);
String token = tokenService.createToken(tokenClaims);
String json = objectMapper.createObjectNode().put("token",
token).toString();
exchange.getResponseSender().send(json);
}
}
```

DataHttpHandler.java

```
package com.iot.http;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.iot.mq.MessageQueueSender;
import com.iot.token.JwtTokenService;
import com.iot.token.TokenParseException;
import com.iot.token.TokenVerificationException;
import io.undertow.server.HttpHandler;
import io.undertow.server.HttpServerExchange;
import io.undertow.util.HeaderValues;
import io.undertow.util.Headers;
import io.undertow.util.Methods;
import io.undertow.util.StatusCodes;
import java.io.IOException;
```

```
import java.util.Map;
import static com.iot.http.AuthenticationHttpHandler.DEVICE_ID;
import static com.iot.http.HttpUtils.*;
import static com.iot.mq.KafkaTopics.DATA_TOPIC;
import static io.undertow.util.StatusCodes.UNAUTHORIZED;
public class DataHttpHandler implements HttpHandler {
public static final String DATA = "data";
private final MessageQueueSender sender;
private final JwtTokenService tokenService;
private final ObjectMapper objectMapper;
public DataHttpHandler(MessageQueueSender sender, JwtTokenService tokenService,
ObjectMapper objectMapper) {
this.sender = sender;
this.tokenService = tokenService;
this.objectMapper = objectMapper;
}
@Override
public void handleRequest(HttpServerExchange exchange) throws Exception {
exchange.getRequestReceiver().receiveFullBytes(this::processMessage,
(exch, e) -> sendServerError(exchange));
}
private void processMessage(HttpServerExchange exchange, byte[] message) {
String authorizationHeader = getAuthorizationHeader(exchange);
if (authorizationHeader == null) {
sendError(exchange, "No 'Authorization' header",
UNAUTHORIZED);
return;
}
Map<String, Object> tokenPayload;
try {
tokenPayload = tokenService.verifyToken(authorizationHeader);
} catch (TokenParseException e) {
sendError(exchange, "Token could not be parsed",
UNAUTHORIZED);
return;
} catch (TokenVerificationException e) {
```

```
sendError(exchange, "Token could not be verified",
UNAUTHORIZED);
return;
}
JsonNode json;
try {
json = objectMapper.readTree(message);
} catch (IOException e) {
sendRequestError(exchange);
return;
}
JsonNode data = json.get(DATA);
String deviceId = getDeviceId(tokenPayload);
if (deviceId == null || data == null) {
sendRequestError(exchange);
return;
}
exchange.dispatch(); // do not end exchange when this method returns,
because saving to kafka is async
sender.send(DATA_TOPIC, deviceId, data.toString())
.thenRun(() -> exchange.getResponseSender().close())
.exceptionally((throwable ->
sendServerError(exchange)));
}
private static String getAuthorizationHeader(HttpServerExchange exchange) {
HeaderValues authorizationHeaderValues =
exchange.getRequestHeaders().get(Headers.AUTHORIZATION);
return authorizationHeaderValues == null ? null :
authorizationHeaderValues.getFirst();
}
private static String getDeviceId(Map<String, Object> tokenPayload) {
if (tokenPayload == null)
return null;
Object o = tokenPayload.get(DEVICE_ID);
if (o instanceof String) {
String deviceId = (String) o;
```

```
return deviceId.isEmpty() ? null : deviceId;
}
return null;
}
}
```

HttpServer.java

```
package com.iot.http;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.iot.auth.AuthenticationService;
import com.iot.auth.MongoAuthenticationService;
import com.iot.mq.KafkaSender;
import com.iot.token.JwtTokenService;
import com.iot.weather.http.ReportHttpHandler;
import io.undertow.Undertow;
import static io.undertow.Handlers.routing;
public class HttpServer {
public static void main(final String[] args) {
JwtTokenService tokenService = new JwtTokenService();
ObjectMapper objectMapper = new ObjectMapper();
AuthenticationService authenticationService = new
MongoAuthenticationService();
DataHttpHandler dataHttpHandler = new DataHttpHandler(new
KafkaSender(), tokenService, objectMapper);
AuthenticationHttpHandler authenticationHttpHandler =
new AuthenticationHttpHandler(authenticationService,
tokenService, objectMapper);
ReportHttpHandler reportHttpHandler = new ReportHttpHandler();
Undertow server = Undertow.builder()
.addHttpListener(8080, "localhost")
.setHandler(routing()
.get("/api/report/device/{id}",
reportHttpHandler)
.post("/api/data", dataHttpHandler)
.get("/api/auth/login/{login}/password/{password}", authenticationHttpHandler))
.build();
server.start();
```

```
}
}
```

DoubleStatsCounter.scala

```
package com.iot.util
import scala.math.sqrt
case class DoubleStats(count: Long, avg: Double, stdDev: Double)
case class DoubleStatsCounter(var n: Long = 0, var sum: Double = 0.0, var
sumOfSquares:
Double = 0.0) {
 def add(number: Double): DoubleStatsCounter = {
 n += 1
 sum += number
 sumOfSquares += number * number
 this
 }
 def merge(anotherCounter: DoubleStatsCounter): DoubleStatsCounter = {
 n += anotherCounter.n
 sum += anotherCounter.sum
 sumOfSquares += anotherCounter.sumOfSquares
 this
 }
 def avg = sum / n
 def stdDev = sqrt(sumOfSquares / n - avg * avg)
 def stats = DoubleStats(n, avg, stdDev)
}
object DoubleStatsCounter {
 def apply(number: Double): DoubleStatsCounter = new
DoubleStatsCounter().add(number)
}
```

WeatherDomain.scala

```
package com.iot.weather
import java.util.Date
import com.iot.util.{DoubleStats, DoubleStatsCounter}
object WeatherDomain {
 // db
 val (keyspace, table) = ("iot", "weather")
 // columns
```

```
 val (deviceId, timestamp, temperatureStats, pressureStats) = ("device_id",
"timestamp", "temperature_stats", "pressure_stats")
 val (count, avg, stdDev) = ("count", "avg", "std_dev")
 case class AggregateWeatherDataRecord(temperatureStats: DoubleStatsCounter,
 pressureStats: DoubleStatsCounter) {
 def merge(anotherRecord: AggregateWeatherDataRecord) = {
 temperatureStats.merge(anotherRecord.temperatureStats)
 pressureStats.merge(anotherRecord.pressureStats)
 this
}
 }
 case class WeatherDatabaseRecord(deviceId: String, timestamp: Date,
 temperatureStats: DoubleStats, pressureStats:
DoubleStats)
 case class WeatherDataRecord(temperature: Double, pressure: Double)
}
ReportHttpHandler.scala
package com.iot.weather.http
import com.datastax.spark.connector._
import com.iot.http.HttpUtils.{extractQueryParameter, sendServerError}
import com.iot.weather.WeatherDomain._
import io.undertow.server.{HttpHandler, HttpServerExchange}
import io.undertow.util.HttpString
import org.apache.spark.{SparkConf, SparkContext}
import org.json4s.NoTypeHints
import org.json4s.jackson.Serialization
import org.json4s.jackson.Serialization._
import scala.concurrent.ExecutionContext.Implicits.global
import scala.util.{Failure, Success}
class ReportHttpHandler extends HttpHandler {
 val conf = new SparkConf()
 .setMaster("local[*]")
 .setAppName(getClass.getSimpleName)
 .set("spark.cassandra.connection.host", "127.0.0.1")
 val sc = new SparkContext(conf)
 implicit val formats = Serialization.formats(NoTypeHints)
```

```
 val corsHeader = new HttpString("Access-Control-Allow-Origin")
 override def handleRequest(exchange: HttpServerExchange): Unit = {
 var weatherTable = sc.cassandraTable[WeatherDatabaseRecord](keyspace, table)
 Option(extractQueryParameter("id", exchange.getQueryParameters))
 .foreach(id => weatherTable = weatherTable.where(s"$deviceId = ?", id))
 exchange.dispatch
 val rowsFuture = weatherTable.collectAsync()
 exchange.getResponseHeaders.add(corsHeader, "*")
 rowsFuture.onComplete {
 case Success(rows) => exchange.getResponseSender.send(write(rows))
 case Failure(e) => sendServerError(exchange)
 }
 }
}
```

WeatherStreamingProcessor.scala

```
package com.iot.weather.stream
import java.util.Date
import com.datastax.spark.connector.cql.CassandraConnector
import com.datastax.spark.connector.streaming._
import com.iot.mq.KafkaTopics.DATA_TOPIC
import com.iot.util.DoubleStatsCounter
import com.iot.weather.WeatherDomain._
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.{SparkConf, SparkContext}
import org.json4s._
import org.json4s.jackson.JsonMethods._
object WeatherStreamingProcessor extends App {
 implicit val formats = DefaultFormats
 val batchDuration = Seconds(1)
 val conf = new SparkConf()
 .setMaster("local[*]")
 .setAppName(getClass.getSimpleName)
 .set("spark.cassandra.connection.host", "127.0.0.1")
 val sc = new SparkContext(conf)
```

```
 val ssc = new StreamingContext(sc, batchDuration)
 type Key = String
 val (zkQuorum, groupId) = ("localhost:2181", "weather-consumer")
 val stats = "stats" // type
 // json fields
 val (temperature, pressure) = ("temperature", "pressure")
 CassandraConnector(conf).withSessionDo { session =>
 session.execute(s"DROP KEYSPACE IF EXISTS $keyspace")
 session.execute(s"CREATE KEYSPACE IF NOT EXISTS $keyspace " +
 s"WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1 }")
 session.execute(s"CREATE TYPE $keyspace.$stats ($count BIGINT, $avg DOUBLE,
$stdDev
DOUBLE)")
 session.execute(
 s"""CREATE TABLE IF NOT EXISTS $keyspace.$table
 ($deviceId TEXT, $timestamp TIMESTAMP,
 $temperatureStats FROZEN<stats>,
 $pressureStats FROZEN<stats>,
 PRIMARY KEY ($deviceId, $timestamp))""")
 session.execute(s"TRUNCATE $keyspace.$table")
 }
 val stream = KafkaUtils.createStream(ssc, zkQuorum, groupId, Map(DATA_TOPIC ->
1),
StorageLevel.MEMORY_ONLY)
 stream
 .map(parseKafkaRecord)
 .map(toAggregateRecord)
.reduceByKeyAndWindow((r1: AggregateWeatherDataRecord, r2:
AggregateWeatherDataRecord) => r1.merge(r2),
 batchDuration, batchDuration)
 .map(toDatabaseRecord)
 .saveToCassandra(keyspace, table)
 ssc.start()
 def parseKafkaRecord(t: (String, String)): (Key, WeatherDataRecord) = {
 val json = parse(t._2)
 (t._1, WeatherDataRecord(
 (json \ temperature).extractOrElse(0.0),
```

```
 (json \ pressure).extractOrElse(0.0)))
 }
 def toAggregateRecord(t: (Key, WeatherDataRecord)): (Key,
AggregateWeatherDataRecord)
=
 (t._1, AggregateWeatherDataRecord(DoubleStatsCounter(t._2.temperature),
DoubleStatsCounter(t._2.pressure)))
 def toDatabaseRecord(t: (Key, AggregateWeatherDataRecord)) =
 WeatherDatabaseRecord(t._1, new Date(), t._2.temperatureStats.stats,
t._2.pressureStats.stats)
```