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
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи інтеграції для приватної хмари”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-23М
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Пятишев В.О.
«____» _____ 2024 р.

Керівник проекту
доктор філософії (PhD)
_____ Дрєєва Г.М.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 123 *"Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерна інженерія"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2024 року

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

Пятишеву Віталію Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи інтеграції для приватної хмари*

2. Керівник роботи *Дрєєва Ганна Миколаївна, доктор філософії (PhD)*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 19-13 від 07.08.2024 року

3. Строк подання студентом роботи до захисту *2.12.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи інтеграції для приватної хмари*

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

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Маркетингове та економічне обґрунтування ІТ-проєкту.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

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

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Доренська А.О.	05.10.2024	14.11.2024
Охорона праці	Марченко К.М., к.т.н., доцент	06.10.2024	16.11.2024

7. Дата видачі завдання « 6 » вересня 2024 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	10.10.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2024 р.	
3.	Розробка моделі компонента	20.10.2024 р.	
4.	Розробка структур даних	25.10.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2024 р.	
6.	Програмування алгоритмів	10.11.2024 р.	
7.	Розрахунок економічної ефективності	13.11.2024 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2024 р.	
9.	Оформлення ПЗ	17.11.2024 р.	
10.	Попередній захист роботи	2.12.2024 р.	

Дата видачі завдання
« 6 » вересня 2024 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2024 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Пятишев В.О. Дослідження та програмна реалізація системи інтеграції для приватної хмари. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи інтеграції для приватної хмари.

Метою розробки є дослідження та програмна реалізація системи інтеграції для приватної хмари.

Об'єктом дослідження є процес інтеграції для приватної хмари.

Предметом дослідження є методи інтеграції для приватної хмари.

Методи дослідження базуються на методах теорії комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи інтеграції для приватної хмари.

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

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

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

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, приватна хмара

ABSTRACT

Pyatyshev V.O. Research and software implementation of an integration system for a private cloud. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this final qualification work for the second (master's) level of higher education, software is developed, which is intended for an integration system for a private cloud.

The purpose of the development is the research and software implementation of the integration system for the private cloud.

The object of research is the integration process for a private cloud.

The subject of research is integration methods for a private cloud.

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

The result of the work is the software implementation of the integration system for the private cloud.

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 a PC with OS Windows 10/11.

The program was developed in the Python environment.

Keywords: computer engineering, private cloud

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БД	–	база даних
ЛОМ	–	локальна обчислювальна мережа
ASP		Active Server Pages – активні серверні сторінки
DHCP	–	Dynamic Host Configuration Protocol – протокол динамичної конфігурації вузла
HTTP	–	HyperText Transfer Protocol – протокол передачі гіпер тексту
IMAP	–	Internet Message Access Protocol – протокол доступу до електроної пошти Інтернету
ICMP	–	Internet Control Message Protocol – міжмережний протокол керуючих повідомлень
MMC	–	Microsoft Management Console
POP3	–	Post Office Protocol Version 3 – протокол поштового відділення, версія 3
SQL	–	Structured Query Language – мова структурованих запросів
SMTP	–	Simple Mail Transfer Protocol – простий протокол передачі пошти
SNMP	–	Simple Network Management Protocol – простий протокол керування мережею
Syslog	–	стандарт відправки повідомлень про зміни які відбуваються в мережі
UDP	–	User Datagram Protocol – протокол користувальницьких дейтаграм

ВСТУП

Актуальність теми. Інтегровані програмно-апаратні комплекси, що включають у себе серверне й мережне встаткування, системи зберігання даних, набір системного й керуючого ПЗ для рішення специфічних завдань, прискорюють уведення в експлуатацію, сприяють передбачуваності й зниженню ризиків, однак підходять вони в основному великим компаніям і провайдерам.

Практично всі провідні вендори пропонують так звані комплексні інтегровані платформи – заздалегідь протестовані програмно-апаратні рішення загального призначення або оптимізовані для виконання тих або інших завдань. У числі прикладів – системи vBlock альянсу VCE, Cisco UCS, HP CloudSystem, IBM PureSystem, Dell Active System, Hitachi Unified Compute Platform (UCP) і ін., а також референсні архітектури спільної розробки декількох вендорів. У ряді випадків подібні рішення позиціонуються як «хмару з коробки» – системи, за допомогою яких замовник може швидко розгорнути інфраструктуру приватної або публічної хмари й впровадити хмарні сервіси.

Інтегровані системи являють собою закінчений програмно-апаратний комплекс для рішення специфічних завдань. Поряд із серверним і мережним устаткуванням, СЗД, набором системного й керуючого ПЗ одним із ключових його компонентів є спеціалізоване прикладне програмне рішення.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи інтеграції для приватної хмари.

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

– Огляд існуючих систем інтеграції для приватної хмари.

– Дослідження системи інтеграції для приватної хмари.

– Програмна реалізація системи інтеграції для приватної хмари.

Об'єктом дослідження є процес інтеграції для приватної хмари.

Предметом дослідження є методи інтеграції для приватної хмари.

Методи дослідження базуються на методах теорії комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод інтеграції для приватної хмари.

– Розроблено вітчизняний продукт інтеграції для приватної хмари, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі інтеграції для приватної хмари.

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.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2024 р.), основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи інтеграції для приватної хмари, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

У чому саме складається відмінність інтегрованої системи від набору встаткування, навіть якщо все це встаткування підтримується одним вендором? І якими характеристиками вона повинна володіти?

До таким ставляться легке розгортання, висока продуктивність і відказостійкість, оптимізація як на апаратному, так і на програмному рівні, швидке виділення ресурсів, єдина система керування й моніторингу, ефективна технічна підтримка й оперативне керування. Убудовані технологічні рішення дозволять ефективніше використовувати ресурси платформи – наприклад, різні типи систем зберігання, шифрування, флеш-технології.

Якщо мова йде про створення хмарної інфраструктури, то інтегрована система повинна містити в собі апаратну частину (сервери, СЗД, мережні ресурси), систему віртуалізації й поверх цього – систему керування хмарними сервісами. Усе компоненти повинні бути протестовані, безперебійно й злагоджено працювати.

Так, наприклад, на створення VCE vBlock і тестування мікрокоду компонентів на взаємну сумісність були витрачені тисячі робочих годин архітекторів і інженерів. Як показує практика, використовуваний багатьма підхід «зроби сам» значно уступає інтегрованим рішенням з погляду часу простою й кількості інцидентів, хоча гнучкість рішення, безумовно, буде вище.

Інтегровані системи повинні бути засновані на спеціально розробленій архітектурі, орієнтованої на конкретні завдання, і гарантувати сумісність апаратних і програмних складових комплексу. Крім того, вони повинні забезпечувати гарну масштабованість ключових компонентів і ємності систем при достатній гнучкості конфігурування комплексу залежно від поточних і

майбутніх потреб замовника. Такі характеристики гарантують необхідна якість платформного рішення і його ефективне використання протягом строку експлуатації.

1.2 Область застосування

До числа головних особливостей інтегрованих систем віднесемо заздалегідь протестовану й веріфіковану вендорами архітектуру, швидке налаштування й запуск в експлуатацію, включення додаткових рішень до складу комплексу (наприклад, системи моніторингу).

З розвитком подібних систем треба було підвищити ефективність використання готових комплексів для виконання різних завдань і забезпечити можливість їхнього спільного застосування з альтернативними технологіями й компонентами, а також адаптації до різних умов роботи. У зв'язку із цим все частіше стали з'являтися конвергентні рішення. Наприклад, Hitachi Unifed Compute Platform, де реалізована концепція уніфікованої платформи, являє собою еволюцію інтегрованих рішень компанії. Ключова особливість UCP – доповнення інтегрованої гетерогенної платформи єдиною системою керування зі стандартного інтерфейсу додатків, що виключає необхідність налаштування різних компонентів системи вручну.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи інтеграції для приватної хмари, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

System Center Virtual Machine Manager 2022

Як створюється хмара за допомогою інструмента – Virtual Machine Manager 2022. Я впевнений що підхід використання хмари як моделі для надання IT-сервісів і ресурсів набирає оберти й багато адміністраторів, та й компанії в цілому, уже добре знайомі з віртуалізацією – і поступово починають переходити на наступний рівень – створення приватної хмари. Ну що ж – давайте більш детально розглянемо цей цікавий процес у дії!

Перш ніж ми перейдемо безпосередньо до розгляду процесу створення хмари в VMM, не зайвим буде ще раз згадати навіщо ми це робимо й у чому буде користь.

Оскільки основою будь-якої сучасної хмари є технологія віртуалізації, то перша асоціація яка повинна виникнути в допитливому розумі – це обчислювальні ресурси. У зв'язку з поширенням віртуалізації, хмара як наступний рівень IT-моделі, що використовує для своїх завдань віртуалізацію, говорить нам про те, що тепер ми вже враховуємо не сервера при обліку потужностей і розгортання сервісів і додатків, а безпосередньо обчислювальні ресурси. Основними ресурсами, які нам необхідні і які ми віртуалізуєм, це ресурси ЦП, ресурси оперативної пам'яті (ОЗП), ресурси сховища для розміщення даних (горезвісний пам'ять) і комунікаційні ресурси (або просто мережі). Звичайно, ми нікуди не дінемося від того, що ВМ і сервіси, що складаються з них, потрібно десь розміщати, і питання розміщення буде звернений безпосередньо до об'єкта комп'ютера, хоста віртуалізації, але з погляду

хмарного підходу нам більше цікавий обсяг ресурсів – адже такі технології як Dynamic Memory не дозволяють уже в стандартному режимі «по-звичці» сприймати ресурси негнучкими блоками, якими хости і є,…

І той факт, що ОЗП розподіляється динамічно, і якщо ми говоримо про надання простору сховищ під ВМ і сервіси – те напевно наткнемося на методи Thin Provisioning і механізми дедуплікації даних, – це лише деякі, але самі явні приклади того, що підхід до споживання (а значить і виділенню) ресурсів у традиційним стилі не дозволяє точно й адекватно дати оцінку ресурсам і правильно їх виділити під сервіси…Тепер ми дивимося на кількість необхідних нам саме обчислювальних і інфраструктурних ресурсів, коли хочемо створити нову ВМ або сервіс на базі ВМ.

Не будемо сильно зациклюватися на деталях хмари, основний момент полягає саме в тому, що, по-перше, ми ставимося до ресурсів як до пулів, наборам, а по-друге – це рівень абстракції, адже віртуалізація абстрагує нас від фізичного рівня на логічний (отут все просто – адже ВМ являє собою ніщо інше, як набір файлів – а це логічний рівень уже), тобто ми оперуємо з віртуальними машинами як з файлами… Для тих хто хоче більш детально розібратися особливо хмари пропоную копати тут. Ну й картинка для більшої ясності.

Відмінні риси хмари як моделі

Отже, давайте переступимо до нашого процесу створення хмари.

Саме таким чином, як на картинці нижче (рисунок 2.1) виглядає консоль керування VMM, – і з першого погляду не дуже зрозуміло, що потрібно робити далі, але давайте разом із цим розберемося.

Машинально так і хочеться залізти у вкладку **Clouds** і почати створювати, але не будемо поспішати. У контексті VMM об'єкт хмари (cloud) є логічним периметром-обмежником, що накладається поверх фізичних ресурсів. А раз мова зайшла про фізичні ресурси, то в першу чергу нам потрібно їх додати, тобто зараз ми додамо хости віртуалізації в VMM – для цього зайдемо в розділ **Fabric**, далі – як на рисунку.

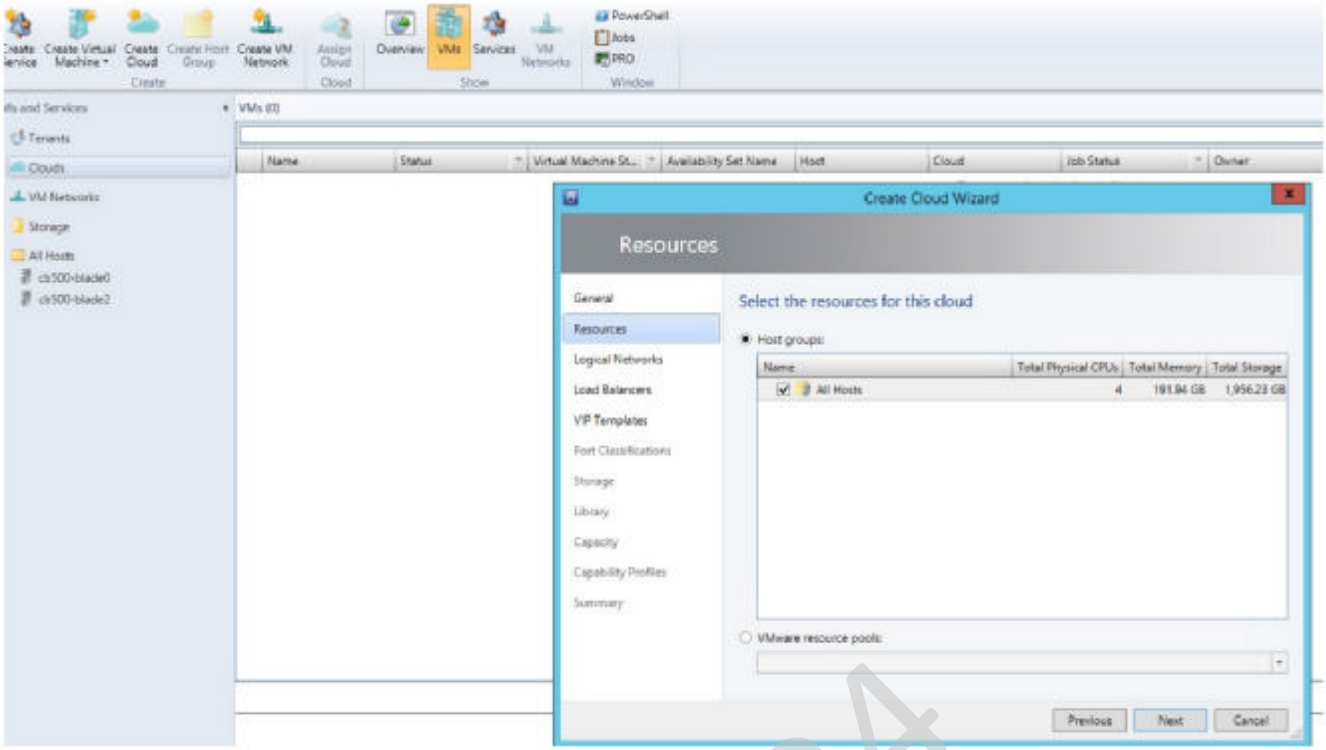

Рисунок 2.1 – Інтерфейс VMM

У якості хостів віртуалізації ми можемо додати безліч різних платформ, не тільки Hyper-V – але й VMware ESXi/vSphere і Citrix XenServer також підтримуються як платформи віртуалізації.

У нашім випадку ми зупинимося на платформі Hyper-V і будемо шукати хости саме із цією технологією. Далі по ходу руху по майстру додавання ресурсів нам потрібно вказати імена цільових хостів – це можуть бути NetBIOS- і FQDN-імена, а також просто IP адреса. Однак, варто відзначити що обліковий запис, під якою ви робите пошук хостів повинна мати права локального адміністратора на цільовому хості віртуалізації – інакше нічого цікавого в нас із вами не вийде. Після того, як майстер знайде потрібні хости – залишається поставити галочки напроти об'єктів, що цікавлять нас, і завершити процес.

Відмінно, ми з вами додали хости віртуалізації й тепер біжимо вже створювати хмару, але…Давайте не будемо поспішати й розберемося, що робити далі. Якщо на хвилинку зупинитися й задуматися, то в результаті наших дій ми

додали 2 хоста (у цьому випадку), у якого є ресурси ЦП, ОЗП (із цим все ровно), але от далі є ще 2 дуже важливих елементи – це сховища, пам'ять і мережі передачі даних. З пам'яттю все зрозуміло: у найпростішому випадку для розміщення ВМ ми можемо використовувати локальні диски нашого сервера, якщо такі є (і в даному прикладі саме така конфігурація й розглядається), але все-таки – у реальному житті ми скоріше будемо працювати із промислової СЗД (яка може підключатися по iSCSI або Fibre Channel, а ще СЗД може бути цілий зоопарк різних – благо в нас SMI-S є на цей випадок.SMI-S – це вендор-нейтральнй протокол керування системами зберігання даних, що дозволяє виділяти й управляти сховищами під ВМ і сервіси прямо в VMM), або як альтернативний варіант, ми можемо використовувати файл-сервера й SMB 3.0-кулі для розміщення ВМ, VMM із цим упорається без проблем.

Якщо з розміщенням навантажень ми в нашім окремому випадку проблему вирішили завдяки локальним дисковим ресурсам наших хостів, то тепер нам варто зайнятися останнім інфраструктурним складовим нашого хмара – мережами

Спершу було б дуже непогано створити єдиний віртуальний комутатор для керування мережею уздовж всіх наших хостів. У контексті VMM такий об'єкт має ім'я **логічного комутатора (logical switch)**, однак крім створення єдиного комутатора, нам також буде потрібно механізм автоматичного призначення мережного адаптера хоста на зв'язок із цим комутатором – а от для цієї мети нам буде потрібно створити **профіль uplink-порту**. З погляду VMM спочатку необхідно створити профіль порту (Для цього **заходимо в Fabric->Networking->Port Profiles і по останньому клікаємо правою кнопкокою миші** – далі без варіантів створюємо профіль).

Зверну вашу увагу на те, що профіль порту може бути створений з урахуванням можливості тімінга (або агрегації інтерфейсів), убудованого в Windows Server 2012/2022. Якщо ви створюєте профіль аплінка на звичайному інтерфейсі, то параметри тімінга ніяк не вплинуть на властивості інтерфейсу.

Якщо ви уважні, то ви звернули увагу на те, що їсти варіант створити не uplink-профіль, а профіль віртуального порту. Профіль віртуального порту, у свою чергу, призначається на віртуальний адаптер ВМ, щоб мапиться на правильний фізичний адаптер – але про цьому не сьогодні, просто для повноти картини завершую розповідь про цю тему.

Тепер давайте перейдемо до створення логічного комутатора (Logical Switch), створюється він всі там же, у вкладці Networking.

Після запуску майстра ми задамо ім'я віртуального свитча-комутатора, ми можемо вибрати розширення комутатора, якщо такі в нас установлені (наприклад, Cisco 1000 Nexus V). Саме ж мнас, що цікавлять, речі перебувають небагато далі**, на вкладці** Uplink – тут ми саме вказуємо створений раніше профіль аплінка.

Профіль віртуального порту нам зараз нецікавий, тому сміло завершуємо налаштування віртуального комутатора на останньому екрані майстра. Після того як ми створили наш віртуальний комутатор з і настроїли профілі, тепер було б непогано застосувати наш комутатор на наші хости Hyper-V. Для цього заходимо в розділ **Fabric->Servers->All Hosts, вибираємо потрібний нам хост і клацаємо по ньому правою кнопкою миші** – далі нас цікавить розділ Virtual Switches (віртуальні комутатори). Далі ми вибираємо New Virtual Switch -> New Logical Switch. Виберіть потрібний мережний адаптер і профіль аплінка. Повторите дану операцію на всіх необхідних хостах (або свисніть PowerShell скрипт і вставте туди все імена).

Тепер справа залишилася за малим. Хости ми зв'язали воєдино, тепер нам потрібно зробити мережу доступної для самої хмари. Тепер нам потрібно створити **логічну мережу (Logical Network)** – єдиний безперервний простір уздовж безлічі хостів, на якому будуть розташовуватися **мережі віртуальних машин (VM Networks)**.

Тут виникає питання: «А в чим різниця між логічною мережею й мережею ВМ?»

Відповідь досить проста: мережі ВМ розміщаються поверх логічних мереж. Тут зміст полягає в тому, що логічна мережа – це безперервний простір мережі уздовж безлічі хостів. Мережі ВМ будуються поверх логічних мереж використовую мережну віртуалізацію, тобто створення ізольованих мереж, які поводяться немов це різні фізичні мережі, тобто вони не знають про існування один одного. У такий спосіб сполучення цих механізмів дозволяє вишиковувати величезна безліч ізольованих віртуальних мереж уздовж хостів, і природно, якщо мережі не знають про існування один одного – те й IP-адреси в них можуть використовуватися повторювані. Саме так і ми надійдемо при створенні нашої мережі. Ще одним важливим моментом є той факт, що хмара як елемент VMM працює з компонентом мережі на рівні логічної мережі, тобто створення логічної мережі є необхідною умовою для створення хмари.

Для включення мережний віртуалізації ми активуємо першу галочку, як на рисунку. Друга ж галочка нам необхідна якщо ми хочемо поставити знак рівності між логічною мережею й мережею ВМ – тобто при цьому дії буде створена мережа ВМ із таким же ім'ям і областю дії як і логічна мережа. Це потрібно більше для мереж керування інфраструктурою, користувальницькі навантаження не рекомендується розміщати таким чином, тому що виникає ризик доступу до інфраструктури з боку неавторизованих користувачів.

Після того, як ми підготували нашу інфраструктуру, настав самий час для того щоб зібрати все компоненти разом і зв'язати їхньою логічною іпостассю об'єкта хмара (cloud) в VMM. для цього **зайдемо в VMs and Services->Clouds->клацання правої кнопки миші й create**. Далі з'явиться майстер створення хмари, що попросить нас задати ім'я хмари, потім попросить вибрати периметр ресурсів (хостів віртуалізації) поверх яких буде натягнута хмара (я вибрав All Hosts), вибираємо потрібну нам логічну мережу, призначаємо сховище під ВМ. Цікавим моментом у створенні хмари є можливість обмежити споживання ресурсів, як у відносному, так і в кількісному вираженні.

Це необхідно для того, щоб користувачі хмари не зжерли всі ресурси, тим самим запобігти недоліку ресурсів для інших користувачів.

Після того, як ми створили хмару залишається призначити цій хмарі його користувачів, які буду як споживати його ресурси (з однієї сторони), так і управляти безпосередньо їм (з іншої сторони). Для цього все в тому же **розділі VMs and Services виберемо створену хмару й натиснемо на клавішу Assign Cloud**. Якщо в нас є створена раніше роль, то ми можемо призначити неї на хмару, але в нашім випадку роль ми ще не створювали – тому давайте розглянемо які варіанти ролей нам доступні й у чому їхній зміст.

І так – на вибір 4 ролі: Fabric Administrator (він же Delegated Administrator, Read-Only Administrator, Tenant Administrator і Application Administrator (a.k.a.Self-Service User). Перша роль – це повноцінний, всі(майже)здатний адмін, але в межах границь хмари, на яке його призначають.Read-Only Administrator – це явно роль горезвісного хелпдеск-траблшутера – тому що ніяких дій крім як простий моніторинг ситуації, ця роль під собою не має на увазі…TenantAdministratorце роль для керування підписками Windows Azure і потрібна для гібридних моделей хмар. Application Administrator – це властиво кінцевий користувач сервісу й споживач ресурсів.

Після того, як ви створили хмару й призначили користувачів, ви можете одержати доступ до хмари з боку користувача й адміністратора на рівні веб-інтерфейсу, для цього необхідно розгорнути App Controller і прив'язати екземпляр сервера VMM – хмари з нього автоматично підтягнуться.

От ми з вами й створили хмару, тепер можно зайнятися питаннями створення ВМ і сервісів у хмарі.

Головні тренди ринку приватних хмар в Україні

Ринок приватних хмар в Україні показує гарну динаміку, за останні два роки на ринку «дозріло» досить багато нових рішень, а проектів стає усе більше, а самі вони – складніше. Найважливішими тенденціями ринку на 2016-2017 роки

є подальша стандартизація послуг, розвиток моделі гібридної хмари, збільшення проникнення рішень із відкритим кодом і, зокрема, OpenStack, і інших.

Одна з головних тенденцій розвитку часток «хмар» пов'язана зі зваженим, поетапним підходом замовників до реалізації цієї концепції. У першу чергу автоматизують тестові середовища розробки нових продуктів і послуг. Далі, якщо дивитися на те, як розвивається західний ринок приватних хмар, треба черга автоматизації керування промисловим середовищем, і, нарешті, третій етап – автоматизація процесу виводу протестованних розробок у промислове середовище, тобто об'єднання двох середовищ або повна автоматизація процесу виділення IT-ресурсів у компанії.

Стандартизація «часток» хмар

Softline відзначає ріст уваги до стандартизованих готових пропозицій частки «хмари» як сервісу від провайдерів або виробників заліза.

З ними солідарні фахівці Fujitsu: З'являються індустріалізовані передналаштовані хмарні рішення, тобто замовники можуть придбати власну хмару максимально задовольняючим вимогам або потребам конкретної сфери діяльності.

Купуючи передналаштоване рішення, замовники мають можливість одержати готову cloud-інфраструктуру у вигляді сервісу від спеціалізованих компаній у дата-центрах провайдерів або на території клієнтів.

IT-інфраструктура надається повністю готової до експлуатації, підтримку й адміністрування здійснює сервіс-провайдер. На неї не буде схованих витрат і складної схеми оплати – всі можливі витрати входять у щомісячний платіж

Вертикалізація ринку

Основною тенденцією на ринку рішень для приватних хмар є цілком певна область їхнього застосування в деяких вертикальних індустріях. Так, наприклад, у фінансовому секторі такою областю найчастіше стають завдання по розробці, а також завдання тестування додатків, як функціонального, так і навантажувального.

Також однієї з тенденцій є індикація найбільш ефективної ситуації для побудови приватного хмарного рішення. Це, як правило, масштабна модернізація корпоративної інфраструктури, або серйозна реорганізація бізнес-процесів, що веде до бізнес-реінжинірингу й зміні структури інформаційних потоків.

OpenStack і відкритий код

Можна виділити кілька основних блоків частки «хмари»: це блок обліку й керування (автоматизації), каталог послуг і інтерфейси користувача й віртуалізація в її широкому змісті (віртуалізація серверів, систем зберігання даних, мережі й іншого). Крім них можуть утримуватися інші менш значні по масштабі модулі.

Для рішення завдання автоматизації роботи IT-служб у режимі «частки» хмари можуть бути використані продукти багатьох відомих виробників, зокрема, HP, Microsoft, VMware,Citrix і ін., але останнім часом все більшу затребуваність одержують рішення на базі відкритого коду (зокрема, OpenStack, Eucalyptus).

Тут необхідно відзначити, що однозначно визнаного лідера серед продуктів автоматизації процесу виділення потужностей поки немає. Кожна компанія самостійно вибирає рішення залежно від потреб сполучення своєї частки «хмари» і публічними ресурсами, побудованими на тих або інших технологіях.

Вага рішень із відкритим кодом для IaaS і часток хмар постійно росте. На думку аналітиків, до кінця 2027 року OpenStack стане де-факто ще одним стандартом IaaS. По суті OpenStack, відкрита платформа, що розвивається співтовариством вендорів, куди входять такі гіганти як Red Hat, IBM, HP, Rackspace, SUSE, безпосередньо OpenStack Foundation, а також VMware, Intel, Samsung і багато хто інші, є втіленням ідеї хмарної рівності для всіх і скасування хмарної рівності.

Прихильники OpenStack затверджують, що вона згодом зможе витиснути повнофункціональні платформи CMP на комерційних підприємствах, хоча поки цього не відбулося. OpenStack зможе згодом стати надійним відкритим ядром

великої й успішної екосистеми комерційних пропозицій. І очевидно, що в цієї платформи більші перспективи, у тому числі на українському ринку.

Гібридні хмари

Гібридна хмара – це сполучення як мінімум однієї приватної хмари й однієї хмарної інфраструктури загального користування, що створює середовище, що забезпечує доступ до хмари й може динамічно наращуватися для керування нерівномірним навантаженням. Таке поле на стику зовнішніх хмарних серверів, яким поки не дуже довіряють і приватних хмар, які вже узвичаюватися. Експерти сходяться в думці, що в майбутньому на українському ринку буде рости сегмент гібридних хмар як найбільш гнучких і які дозволяють оперативно управляти ресурсами й витратами.

П'ять міфів про гібридні хмари, які варто знати

Міф 1: Приватне + Публічне = Гібридне

Примітивне сполучення інфраструктури приватної й публічної хмар не дасть вам справжня гібридна хмара. Більше того, у підсумку ви можете одержати весь набір ризиків і недоліків обох рішень, не отримавши ніяких переваг. Переміщаючи захищені дані в публічну хмару можна легко порушити закони про захист даних. У той же час міграція додатків з публічного в приватну хмару може привести до несподіваних витрат. Справжня гібридна хмара – це розподіл робочого навантаження й керування накопичувальними й мережними ресурсами, що допомагає скоротити ризики й збільшити продуктивність.

Міф 2: Гібридні хмари складно використовувати й важко налаштовувати

Якщо використовувати повністю пророблені рішення, можна спростити використання гібридних хмар. При цьому залишається можливість вибору: технології стандартизації гібридної хмари (VMware, Microsoft, OpenStack) і компаній, що надають доступ до публічних хмар, з якими можуть взаємодіяти приватні хмари. Професійно зроблене рішення дозволяє спростити три основних елементи розробки:

– комплексні інтеграцію й тестування, що дозволяють переконатися в тому, що все компоненти працюють, як покладено;

– використання конвергентної інфраструктури, що значно спрощує забезпечення й розгортання рішень; і

– заздалегідь задані схеми для служб із робочими потоками, щоб автоматизувати ініціалізацію й виділення ресурсів через портал самообслуговування.

Міф 3: Публічна хмара – найбільше економічно вигідне рішення

Насправді, якщо взяти до уваги завдання керування, ризики й необхідність відповідності законодавчим вимогам, то модель гібридної хмари виявляється більше дешевої в експлуатації. Занадто просто порушити локальні або глобальні правила захисту, переміщаючи дані або завдання в публічну хмару. Закони й місцеві вимоги розрізняються на різних ринках, і деякі з них настільки заплутані, що компанії про всякий випадок повністю уникають публічних хмар. Отже, скоріше всього ви зволієте використовувати приватну хмару для якихось конфіденційних завдань. У Німеччині, наприклад, дуже тверді правила зберігання й обробки даних. Оптимальним рішенням є використання публічної й приватної хмар, що гармонійно працюють разом і необхідні переваги, що забезпечує всі.

Міф 4: Поміщаючи дані в хмару, ви втрачаєте контроль над ними

У той час як найбільш агресивні постачальники хмарних послуг можуть ускладнити процедуру одержання або міграції ваших даних, правильно організоване гібридна хмара дозволяє вам зберегти контроль над ними. Правильно настроєна гібридна хмара здатна гарантувати швидкий доступ до публічних і приватних ресурсів, забезпечуючи при цьому як контроль і прозорість процесів для IT-відділів, так і можливість самообслуговування на вимогу розроблювачів і користувачів додатків.

Міф 5: Це дуже непросто – зрозуміти, які додатки повинні перебувати в хмарі

Організації дуже часто страждають як від надмірно ускладненої IT-архітектури, так і від того, що не мають повного переліку своїх IT активів у зв'язуванні з бізнес-додаткам. Маючи на руках тільки таблицю й не маючи належну методологію, практично неможливо точно оцінити, де можна використовувати додаток, не говорячи вже про те, у яку хмарну архітектуру його розмістити. У свою чергу, експерти, що використовують автоматизовані платформи для збору й аналізу даних, можуть надати розгорнутий звіт про роботу додатка, і допомогти вам визначитися з подальшими діями: перемістити той або інший додаток, консолідувати, модернізувати або взагалі відмовитися від його використання.

Керування життєвим циклом хмари

Крім віртуалізації й централізованого керування всіма компонентами рішення актуальними сьогодні можна вважати технології по керуванню повним життєвим циклом систем, забезпеченню відказостійкості й балансування навантаження, можливість надання послуг із самообслуговування, розширена звітність і інтеграція із системами біллинг.

Крім того, клієнти будуть усе більше зацікавлені в тому, щоб одержувати весь комплекс послуг (не тільки хмарних, але й у цілому пов'язаних з роботою IT) від одного провайдеру, за принципом «одного вікна». Переваги цього підходу зрозуміле: спрощення адміністрування, індивідуальний підхід до составів послуг, зниження вартості кожної послуги за рахунок їх «пакетної» закупівлі, централізація відповідальності, забезпечення єдиного рівня сервісу для підрозділів у будь-яких регіонах – це особливо важливо компаніям.

Розвиток систем безпеки

Динаміка розвитку ринку буде залежати від того, наскільки безпечними стануть рішення, надавані провайдерами. Відразу кілька епізодів, пов'язаних з «проривом оборони» великих користувальницьких ресурсів знову привернули

увагу користувачів і аналітиків до питань інформаційної безпеки. Цей сегмент IT-галузі буде найближчим часом розвиватися не менш динамічно, чим хмарні технології, він тісно з ними зв'язаний.

В 2023 році українські компанії й приватні користувачі витратили на програмне забезпечення для захисту інформації на 9,2% більше, ніж роком раніше. При цьому більше 50% ринку рішення для інформаційної безпеки займає ПЗ для захисту кінцевих пристроїв. Нерідко особисті файли користувача розміщені в публічній хмарі, а робітники – у корпоративній частці, а доступ до цих ресурсів здійснюється з того самого пристрою. За 1 квартал 2024 року було заблоковано 1,13 млрд атак на комп'ютери й мобільні пристрої користувачів, а також 353 млн онлайн-атак.

І приватним користувачам, і компаніям необхідно вчитися новим правилам життя у світі, де практично не існує секретів. Провайдери і їхні клієнти будуть виробляти правила розумного компромісу й спільного захисту від хакерів.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – динамічна інтерпретована об’єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Офіційний сайт мови програмування Python https://www.python.org/. Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові. Python – багатоплатформова мова програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних,

для побудови графічного інтерфейсу програм тощо. Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Python абсолютно безкоштовний.

Швидкість виконання коду Python

Один з можливих недоліків Python – швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C.

Втім, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості застосунків швидкість розробки важливіша швидкості виконання, а програми на Python зазвичай пишуться набагато швидше.

Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор.

Використання Python

Python використовується для різних цілей: для створення ігор і веб-застосунків, розробки внутрішніх інструментів для різноманітних проектів. Мова також широко застосовується в науковій області для досліджень і розв'язування прикладних завдань.

Застосування мови програмування Python:

1. BitTorrent – протокол для обміну даними.

2. Ubuntu Software Center – вільне програмне забезпечення для пошуку, установки і видалення пакунків в системі Ubuntu Linux.

3. Blender – програма для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, пост-обробки відео, а також створення відеоігор.

4. GIMP – растровий графічний редактор, із підтримкою векторної графіки.

5. World of Tanks.

6. Вільна енциклопедія Вікіпедія.

7. Пошукова система Google.

8. DropBox – файловий хостинг, що включає персональне хмарне сховище, синхронізацію файлів і програму-клієнт.

9. YouTube – популярне відеосховище.

10. ...

Версії Python

Мови програмування з часом змінюються – розробники додають в них нові можливості, а також виправляють помилки. Так з'являються різні версії мови. Наприклад, код написаний на Python 2 у більшості випадків не буде працювати у версії Python 3 без внесення додаткових змін.

Процесор є найважливішим компонентом в комп'ютері. Одна з основних функцій процесора – це обробка даних згідно комп'ютерної програми, яка є списком інструкцій, шляхом виконання арифметичних і логічних операцій над фрагментами даних.

Кожна інструкція в програмі – це команда, яка «повідомляє» процесору, яку операцію він повинен виконати. Процесор комп'ютера може розуміти лише ті інструкції, які написані на машинній мові. Машинна мова – це штучна мова, створена для передачі команд комп'ютеру. За допомогою машинної мови створюються ефективні програми, оскільки розробник отримує доступ до всіх можливостей процесора. Машинна мова – мова низького рівня.

Інструкція машинної мови існує для кожної операції, яку процесор здатний виконати – є інструкція для додавання чисел, є інструкція для віднімання чисел і т.д. Увесь набір інструкцій, який центральний процесор може виконати, відомий як набір інструкцій процесора.

Наприклад, у вас є певна програма, яка зберігається на диску вашого комп'ютера. Для виконання програми, ви здійснюєте подвійний клік на значку програми. Це змушує програму копіюватися з диска в оперативну пам'ять, після

чого процесор комп'ютера виконує копію програми, яка знаходиться в оперативній пам'яті.

Коли процесор виконує інструкції програми, він бере участь у процесі, який є відомим як цикл fetch – decode – execute (отримати – декодувати – виконати). Цей цикл виконується для кожної інструкції у програмі і складається з трьох кроків:

Отримати

Програма – це послідовність інструкцій на машинній мові. Першим кроком циклу є завантаження (отримання) наступної інструкції з пам'яті в процесор.

Декодувати

Інструкція машинної мови – це двійкове число, яке представляє команду, що повідомляє процесору виконати певну операцію. На цьому кроці процесор декодує інструкцію, яку було «витягнуто» з пам'яті, для визначення того, яка операція повинна виконуватись.

Виконати

Останній крок циклу – виконати операцію.

Хоча процесор комп'ютера розуміє тільки машинну мову, людині непрактично писати програми на машинній мові. Така програма може мати тисячі або навіть мільйони бінарних інструкцій, і написання такої програми буде дуже обтяжливим процесом.

З цієї причини була створена мова асемблера як альтернатива машинній мові. Замість використання двійкових чисел для написання інструкцій, мова асемблера використовує короткі слова, відомі як мнемокоди.

Незважаючи на те, що мова асемблера не вимагає двійкових інструкцій, як у випадку машинної мови, проте вона вимагає високих знань про процесор. Використовуючи мову асемблера, навіть для найпростішої програми, необхідно написати велику кількість інструкцій.

Мова програмування високого рівня дозволяє створювати складні програми, не знаючи, як працює процесор, і не записуючи великої кількості інструкцій низького рівня. Крім того, більшість мов програмування високого рівня використовують слова, які легко зрозуміти.

Python – одна із популярних сучасних мов програмування високого рівня. Python – інтерпретована мова програмування. Python – це високорівнева інтерпретована мова програмування, на відміну від C++, яка є прикладом компільованої мови програмування. Назва Python відноситься як до мови програмування, так і до інтерпретатора – комп'ютерної програми, яка зчитує початковий код (написаний на Python) і виконує інструкції (команди).

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

1. Компілятор.

2. Інтерпретатор.

Завантаження Python

Версії інтерпретатора Python для різних операційних систем доступні для безкоштовного завантаження за адресою https://www.python.org/downloads.

Середовище програмування для Python

Для написання програм використовують текстові редактори або інтегровані середовища розробки, які включають в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо.

Текстові редактори та інтегровані середовища програмування для Python:

– IDLE – стандартний редактор Python. Встановлюється разом з Python для користувачів Windows, окремим пакунком для користувачів Linux.

– Notepad++ – безкоштовний текстовий редактор початкового коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

– Visual Studio Code – це легкий, але потужний редактор початкового коду, який розповсюджується безкоштовно і доступний у версіях для платформ Linux, Windows і macOS.

– PyScripter – інтегроване середовище розробки для мови програмування Python. Для користувачів Windows. Поширюється безкоштовно.

– Wing IDE 101 – вільне інтегроване середовище для Python, розроблене для навчання програмістів-початківців. Для користувачів Linux, Windows і macOS. Поширюється безкоштовно.

– Geany – вільний текстовий редактор з базовими елементами інтегрованого середовища розробки, доступний для операційних систем Linux, Windows і macOS.

– PyCharm – інтегроване середовище розробки для мови програмування Python. PyCharm є власницьким програмним забезпеченням. Наявна безкоштовна версія Community з усіченим набором можливостей. Для користувачів Linux, Windows і macOS.

– Thonny – IDE для вивчення програмування мовою Python. Для користувачів Linux, Windows і macOS.

– Mu – редактор коду Python для програмістів-початківців. Для користувачів Linux, Windows і macOS.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи інтеграції для приватної хмари.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Усе більше підприємств по усьому світі вибирають хмарну архітектуру й, в основному, по двох причинах. Вона робить бізнес більше динамічним, дозволяє швидко реалізувати нові бізнес-моделі, а також сприяє ефективному використанню IT-ресурсів, спрощує експлуатацію IT-інфраструктури. Однак перехід у хмару вимагає аналізу зв'язку IT з бізнес-стратегією й припускає не тільки з технологічні, але й організаційні зміни, реорганізацію бізнес-процесів.

Якщо ви ще не готові до публічної хмари, або це суперечить корпоративній політиці, то можна одержати значні переваги від захищеної приватної хмари. Розгортаючи хмару на своїй площадці, ви забезпечите кращий контроль над витратами й засобами безпеки, разом з тим домігшись більшої гнучкості бізнесу й економії операційних витрат, властивих хмарі. За даними дослідження Oracle, більше половини (55%) респондентів відзначають, що вони досягнуть «хмарної зрілості» в освоєнні приватних хмар у найближчі два роки.

Життєвий цикл хмари містить у собі планування IT-архітектури й варіантів консолідації додатків, створення хмарної інфраструктури, підготовку сервісів, їхнє тестування й публікацію для користувачів, моніторинг і керування хмарною інфраструктурою, тарифікацію й біллинг, оптимізацію ресурсів. Як же побудувати приватна хмара? Можна виділити наступні кроки.

Крок 1. Планування хмари

Приватна хмара припускає віртуалізацію, стандартизацію й автоматизацію. Хоча віртуалізація не обов'язково лежить в основі приватної хмари, це найпоширеніший підхід. Віртуальне середовище, як правило, вимагає централізації ресурсів зберігання даних. У сучасних ЦОД вона охоплює всі рівні – від обчислювальних до мережних ресурсів.

При проектуванні хмари необхідна інвентаризація IT-активів і документування процесів, ревізія корпоративних додатків, аналіз доцільності й способів перекладу додатків у хмару. Щоб консолідувати поточні додатки, треба проаналізувати, як розподіляється навантаження. Для рішення цього завдання використовуються спеціальні інструменти.

Перехід до приватної хмари припускає аналіз готовності IT-інфраструктури й при необхідності – її модернізацію й стандартизацію, включаючи системи керування й забезпечення безпеки. Одна із цілей створення приватної хмари – автоматизація, але її складно досягти без стандартизації IT-систем.

Крім того, необхідні побудова або зміна процесів керування IT відповідно до хмарної моделі. Потрібно продумати, як буде відбуватися розгортання хмари й міграція в нього IT-сервісів, які організаційні зміни для цього необхідні. Таким чином, на даному етапі відбувається детальне пророблення організаційної й технічної складової міграції.

Крок 2. Створення інфраструктури

Для хмари необхідно вибрати технологічну платформу. Вона повинна відповідати характеристикам і принципам майбутньої інфраструктури. Правильно побудована архітектура визначає, які типи сервісів і наскільки гнучко можна буде надавати користувачам.

Приватні хмари можуть бути реалізовані на різних апаратних платформах залежно від розміру підприємства, обсягу й класу розв'язуваних завдань – від ЦОД для великих компаній до декількох серверів x86 для SMB.

У цей час вендори пропонують системи, спеціально спроектовані для реалізації частки хмари. Інтегровані програмно-апаратні комплекси, що включають у себе серверне й мережне встаткування, СЗД, системне й керуюче ПЗ для рішення специфічних завдань, сприяють передбачуваності результату й зниженню ризиків.

Такі комплексні рішення призначені для критично важливих завдань, здатні до масштабування, характеризуються високою продуктивністю й простотою керування. Їх можна використовувати для консолідації різнотипних навантажень. Існують і спеціалізовані рішення, які створені під конкретні завдання, наприклад, обробку більших обсягів даних.

За прогнозом Oracle, в 2018 році багато підприємств сфокусуються на використанні предінтегрованої IT-інфраструктури, а не на спробах «самоскладання», а IT-команди зрозуміють, що «один розмір» не може підійти для всього. Вони розділять робочі навантаження на кілька напрямків і будуть намагатися оптимізувати кожне окремо, а не всі разом.

Спеціалізоване ПЗ дозволяє реалізувати функціональні атрибути хмари: сервісну модель, автоматизоване обслуговування споживачів, масштабованість і моніторинг ресурсів, динамічний розподіл навантаження.

Крок 3. Налаштування

Серед переваг інтегрованих платформ – простота розгортання, гарантована працездатність, простота керування. Виробник ретельно контролює все компоненти, тому рішення працює більш стабільно, демонструє високу продуктивність, відрізняється простотою використання й мінімумом простоїв.

Побудова хмарної інфраструктури зі стандартних компонентів і її налаштування дають ряд переваг: вирішуються проблеми із сумісністю, стандартні елементи інфраструктури мають відомі характеристики продуктивності, а ресурси, необхідні для роботи сервісів, можуть бути визначені заздалегідь шляхом тестування.

Етап налаштування включає підготовку віртуальних машин, БД, серверів додатків, призначення прав користувачам. Хмара також має на увазі стандартизацію створюваних об'єктів на основі механізму шаблонів.

Крок 4. Тестування

Далі проводиться тестування хмари. Отримані результати дозволять скласти чітке уявлення про те, як той або інший сервіс або додаток «почувають»

себе в хмарі. Вони допоможуть виявити слабкі місця обраної архітектури й усунути їх уже на цьому етапі.

Складні технологічні рішення мають на увазі кілька типів тестів – тестування модулів для аналізу компонентів рішення, функціональне тестування, тобто перевірку того, що продукти й компоненти працюють саме так, як планувалося, а також інтеграційне тестування рішення як єдиного цілого. У системах з високими вимогами до безпеки проводяться тести на захист від несанкціонованого доступу.

Комплексні інтегровані платформи, що використовують стандартизовану архітектуру, не вимагають глибокого додаткового тестування, що дозволяє істотно скоротити час уведення системи в експлуатацію.

Крок 5. Замовлення сервісів

Хмара з його автоматизацією припускає замовлення й роботу із сервісами кінцевих користувачів за допомогою порталу самообслуговування. Створення порталу самообслуговування спрощує доступ користувачів до необхідних сервісів і ресурсів IT, і, в остаточному підсумку, розвантажує фахівців першої лінії підтримки. Він надає можливості ефективного самообслуговування різних груп споживачів за рахунок автоматизованого надання IT-послуги «за запитом».

За допомогою порталу самообслуговування користувач може створити заявку, у якій у формалізованому виді вказати вимоги до IT-ресурсів, необхідним для нового сервісу. Заявка або перевіряється адміністратором хмари, або, при відповідних налаштуваннях, виконується системою керування хмари автоматично.

Крок 6. Моніторинг

Обов'язковий компонент хмари – система моніторингу й керування. Зокрема, вона дозволяє контролювати продуктивність додатків, швидко реагувати на проблеми. Необхідно також контролювати використання дискового простору, оперативній пам'яті, обчислювальній потужності й заздалегідь передбачати вичерпання цих ресурсів. Особливо зручні універсальні засоби

моніторингу й керування – від диска до додатка й хмари. Результатом повинне стати підвищення якості обслуговування користувачів і скорочення експлуатаційних витрат завдяки ранньому усуненню несправностей. Комплексний моніторинг підвищує надійність IT-інфраструктури в цілому – скорочується час простою. Крім того, завдяки системі моніторингу не допускається безконтрольний ріст числа віртуальних машин, з'являється можливість обґрунтовувати й планувати закупівлю IT-систем. При необхідності можна організувати біллинг використання обчислювальних ресурсів.

Крок 7. Тарифікація й біллинг

Одна з важливих характеристик хмари – можливість оплати тільки по факті використання ресурсів, поява простого й прозорого способу розрахунку собівартості IT-послуг. Інтегровані в приватна хмара інструментальні засоби дозволяють будувати складні плани біллингу, що враховують десятки характеристик використання встаткування й ПЗ.

Потужні системи тарифікації й біллингу допомагають сформувати гнучкий план оплати ресурсів. При використанні хмарних сервісів для кожного об'єкта на основі тарифного плану будуть формуватися звіти про платежі. Вони можуть бути основою реальної оплати (у сполученні з біллинговою системою) або контролю споживаних ресурсів і для взаєморозрахунків.

Крок 8. Оптимізація

Прості засоби створення хмарної інфраструктури дозволяють розгорнути приватну хмару за один-два тижні, організувати його експлуатацію. Керування всім технологічним стеком хмари від устаткування до додатків здійснюється централізовано.

При впровадженні приватної хмари підвищується оперативність роботи IT-відділу – у будь-який момент по запиті бізнесу він може розгорнути потрібний сервіс за 5-10 хвилин. IT-фахівці лише піднімають віртуальну машину із шаблона й установлюють необхідний сервіс, у той час як компанії із традиційної IT-інфраструктурою довелося б замовляти сервер, установлювати на нього

програмне забезпечення й додатки, підключати його до мережі передачі даних і інфраструктурі зберігання.

Приватна хмара – зручна й ефективна інфраструктура, що має більшу гнучкість і ефективністю, чим традиційні IT. Його характеризують швидкість перебудови, масштабованість і ефективність. Для багатьох організацій це природний перший крок до освоєння хмарних технологій. Крім того, використання такої хмарної моделі служить базою для створення гібридної хмари.

Як правило, рішення для приватної хмари – мультивендорні, і технологічний стік у публічній і приватній хмарі не ідентичний. Деякі постачальники можуть запропонувати для приватної й публічної хмари ті самі платформи, що заважає використовувати потенціал вільної переносимості робітників навантажень між приватним і публічним хмарним середовищем. Але ж було б зручно, якби розгортання хмарних рішень в інфраструктурі клієнта виглядало точно так само, як ще одна дата-центр у публічній хмарі. Ви могли б розширювати ресурси свого дата-центра за рахунок публічної хмари або почати розробку додатка в публічній хмарі, а потім запустити його роботу у своєму даті-центрі. Сьогодні це стає реальністю.

3.2 Розробка структурної схеми

У чому відмінність конвергентних і інтегрованих комплексних платформ? Часто між ними не роблять розходжень, але можна спробувати розділити ці поняття.

Конвергентні рішення виникають на стику традиційних технологій, що ставляться до різного галузям. Вони виходять за межі традиційного підходу й розширюють границі використання традиційних рішень. Інтегровані системи, що складаються з різних елементів апаратно-програмного забезпечення, супроводжуються відповідними сервісами й вирішують конкретне завдання або

певний клас завдань. Але, що характерно, всі елементи такої системи є традиційними для даної галузі (наприклад, сервери, системи зберігання даних, мережне встаткування, програмне забезпечення) і несуть із собою тільки властиві їй переваги. Як правило, у них немає чогось революційного, що істотно міняло б підхід до рішення завдання.

Під конвергентною інфраструктурою розуміють набір технологій і продуктів, об'єднаних в одне рішення. У більшості платформ для побудови приватних хмар конвергенція існує на рівні мереж. Використовуване спеціалізоване комутаційне встаткування поєднує різні типи підключень, у тому числі високошвидкісні. Крім цього, воно оптимізує роботу додатків замовника.

Інтегроване рішення націлене на досягнення максимальної продуктивності при мінімальній вартості у випадку виконання вузькоспеціалізованого завдання. Конвергентне рішення орієнтоване в основному на якнайшвидше одержання потрібного результату при мінімальних ризиках і претендує на універсальність. Крім того, воно вигідно наявністю гарантованих характеристик, мінімальними ризиками впровадження й можливістю висновку єдиного сервісного контракту.

Інтегровані системи, як правило, складаються з апаратних систем стандартної архітектури й необхідного набору програмного забезпечення для виконання спеціалізованих завдань, що дає ряд переваг. Усе компоненти перевірені, протестовані й підібрані таким чином, щоб обчислювальні ресурси використовувалися з максимальною ефективністю. З фінансової точки зору їхнє застосування доцільно саме при рішенні тих завдань, для яких вони й створювалися.

Серед переваг інтегрованих платформ – простота розгортання, гарантована працездатність (тобто мінімальні ризики впровадження завдяки попередньому тестуванню), простота керування (відповідна функціональність теж інтегрована). До числа переваг комплексних інтегрованих платформ відносимо стандартизовану архітектуру й дизайн – вони не вимагають глибокого додаткового тестування, що дозволяє істотно скоротити час уведення системи в

експлуатацію. Виробник ретельно контролює все компоненти, тому рішення працює більш стабільно, демонструє високу продуктивність, відрізняється простотою використання й мінімумом простоїв (як запланованих, так і пов'язаних з аваріями. Інтегровані системи – це гарантована відказостійкість і масштабованість, відповідальність вендора за всю платформу, підтримка новітніх версій системного ПЗ.

Готові інтегровані платформи називають рішеннями «з однієї коробки». Замовник не витрачає час на пошук і перевірку сумісності компонентів і додатків, на замовлення серверів, систем зберігання даних, мережного встаткування, програмного забезпечення для віртуалізації й моніторингу. Все це вже протестовано й перевірено на сумісність. Вендор гарантує безшовну інтеграцію всі складові платформи і її працездатність як єдиного комплексу. Одне з важливих переваг подібних систем – єдина крапка обігу для технічної підтримки».

Інтегровані платформи відрізняють простота й можливість швидкого розгортання. Готова система забезпечує передбачувану продуктивність відповідно до вимог кінцевого завдання. Застосування рішень на базі інтегрованих платформ у максимальному ступені рятує від проблем сумісності, масштабованості, модернізації й подальшої підтримки. Таким чином, знижується вартість розробки, впровадження й підтримки комплексу протягом усього строку експлуатації.

Безсумнівна а таких рішень – можливість впроваджувати їх в інфраструктуру підприємства з мінімальними витратами. Причому вже на етапі закупівлі інтегрованої платформи замовник визначає її можливості, співвідносячись їх зі своїми вимогами. Але якщо останні були розраховані погано, ціна помилки буде дуже висока. Саме тому варто звертати особливу увагу на вихідні вимоги й ретельно пророблять їх ще задовго до вибору на користь того або іншого рішення.

Нерідко вендори пропонують спеціалізовані системи – інтегровані платформи, оптимізовані для роботи з певним програмним забезпеченням. Серед прикладів – системи Oracle Exalogic, Exadata і Exalytics, IBM PureData або PureApplication. Такі платформи дозволяють домогтися більше високої продуктивності в порівнянні з універсальними. Якщо перед компанією коштує конкретне завдання, наприклад розгортання бази даних для надання оперативного й безперебійного доступу до інформації, то використання спеціалізованої платформи буде найкращим рішенням. Оскільки вимоги до інфраструктури в різних додатків помітно відрізняються, такі рішення, оптимізовані для конкретних IT-сервісів, – зручний для замовника варіант. Однак чим більше уніфікованої є платформа, тим легше її обслуговувати й підтримувати, а розмаїтість апаратних і програмних засобів приводить до більше частого виникнення проблем і ускладненню процесу експлуатації рішення.

Від комплексних інтегрованих платформ, призначених для побудови приватної хмари (VCE vBlock, HP CloudSystem, IBM PureSystems і т.д.), варто відокремлювати спеціалізовані програмно-апаратні комплекси, призначені для обробки великого обсягу даних (Oracle Exadata, IBM Netezza і ін.). Останні створені під цілком конкретні завдання – для обробки більших обсягів даних – і при правильному використанні є найбільш оптимальним рішенням.

Спеціалізовані системи дозволяють домогтися істотного підвищення продуктивності. Наприклад, рішення на зразок Oracle Exadata, IBM Netezza, SAP HANA підвищують швидкість обробки даних у десятки, а в деяких випадках і в сотні разів. Крім того, при необхідності продуктивність комплексу легко збільшити за рахунок нарощування апаратних ресурсів.

Універсальна інтегрована платформа не завжди здатна вирішити конкретне завдання. Вузькоспеціалізовані завдання, наприклад обробка «важких» баз даних або надання віртуальних робочих місць для більше десятка тисяч користувачів, вимагають оптимізованих рішень

Система повинна призначатися для конкретного завдання, але при цьому бути масштабованої. Навіть у випадку вибору «коробкових» рішень, а не окремих компонентів потрібно сформулювати цілком певні вимоги.

Використання спеціалізованої платформи має сенс, якщо є чітко обкреслене завдання й потрібно оптимізувати під неї платформу – по продуктивності, співвідношенню ціна/якість або інші параметри. У такому випадку можна «вичавити» з її максимум, але прийде поступитися гнучкістю. А відсутність гнучкості має на увазі, що із часом завдання не будуть змінюватися або, принаймні, зміни будуть незначні. Втративши в гнучкості, можна істотно виграти в ціні й продуктивності (у тому числі за рахунок відмови від додаткового функціонала і його підтримки).

Однак у багатьох випадках, коли мова йде про центри обробки даних, обслуговуючих одне або кілька підприємств, доцільніше використовувати універсальні платформи, які дозволять виконувати різні завдання й використовувати різні підходи, тому універсальність більше затребувана. Але це не означає, що спеціалізовані системи не потрібні – вони займають свою нішу на ринку.

Вибір на користь спеціалізованої платформи виправданий, якщо протягом як мінімум трьох років на цій платформі планується використовувати тільки певний додаток. Спеціалізовані платформи доцільніше застосовувати, коли завдання вимагає більших обчислювальних ресурсів і нема сенсу віртуалізувати додаток. До того ж управляти такою системою набагато легше, ніж самостійно зібраної з різних компонентів.

Спеціалізовані рішення можуть забезпечити більше високі показники продуктивності й ефективності для конкретного додатка, але це чревате певними негативними наслідками: виникненням ефекту «прив'язки» до платформи й необхідністю оптимізувати сам додаток, неможливістю альтернативного використання такої платформи, залежністю від програмного забезпечення, підтримуваного платформою, труднощами переходу на нові версії платформи й

т.д. В остаточному підсумку питання зводиться до стратегічного вибору платформи й оцінці ризиків експлуатації. Досвід компанії Hitachi показує, що ринок усе більше відмінюється до вибору рішень на базі відкритих універсальних платформ. Їхня продуктивність і ефективність здатні задовольнити самих вимогливих замовників, при цьому вибір версії й типу програмної платформи не обмежується

Очевидних недоліків у таких рішень ні, але є «ціна питання. Чи готові ви платити за «перевірені на сумісність» компонента, якщо в переважній більшості випадків вони являють собою стандартні інтерфейси й компоненти? Або за «рекомендовані конфігурації», якщо система однаково буде масштабуватися відповідно до конкретних вимог? До того ж у хмарній інфраструктурі ми максимально абстрагуємося від фізичної складової й оперуємо такими поняттями, як віртуальна машина, а не фізичний сервер, профіль зберігання, а не дисковий тім, мережні сервіси, а не мережні пристрої.

Вартість системи збільшується за рахунок надмірності, взаємної інтеграції компонентів, їхнього тестування на сумісність і комплексну підтримку. Подібні платформи може собі дозволити далеко не кожне підприємство.

Недостатня гнучкість. Такі платформи не відрізняються великою гнучкістю. Далеко не всі можливі сценарії можна реалізувати із залученням інтегрованої платформи. Іноді це може бути пов'язане з масштабами системи. Наприклад, платформа розрахована на підтримку 200 або 500 віртуальних машин, а підприємству потрібне рішення рівне на 300 ВМ. Якщо ж замовник збирає системи самостійно, у нього з'являється можливість настроїти їх відповідно до власних потреб.

Надмірність. Складності при впровадженні комплексних інтегрованих платформ найчастіше виникають у зв'язку зі спробами застосувати такі рішення для малих конфігурацій. Найчастіше навіть у мінімальній конфігурації забезпечується значно більша продуктивність, чим це необхідно для виконання окремого завдання, або такі вимоги регулярно міняються. Для подібних випадків

передбачаються різні сценарії впровадження комплексних інтегрованих платформ. Рішення Hitachi UCP, зокрема, дозволяють задіяти інтегровану платформу для одночасного виконання різних завдань і динамічно перерозподіляти між ними необхідні ресурси. Тим самим підвищується ефективність використання платформи й забезпечується захист інвестицій.

Проблеми інтеграції з успадкованою інфраструктурою. Далеко не всі завдання ефективно зважуються на інтегрованих платформах. Припустимо, підприємство хоче перевести всі свої бізнес-процеси в приватна хмара. При цьому виникає питання: що робити зі старим «залізом»? Найчастіше вендор не гарантує сумісності вже наявного в замовника встаткування й ПЗ приватної хмари, особливо якщо мова йде про апаратне забезпечення іншого виробника. Але сьогодні вже існують рішення, які дозволяють будувати приватна хмара на наявному в замовника встаткуванні.

Комплексна інтегрована платформа – закінчений продукт, що поєднує в собі й серверній платформі, і мережні модулі, і системи зберігання. Завдяки наявності уніфікованих компонентів такий комплекс досить добре конфігурується і їм легко управляти, що є безсумнівним плюсом при створенні інфраструктури з нуля. Але інтегровану платформу не можна назвати універсальної – її архітектура опирається на рішення конкретних вендорів, а виходить, інтеграція із уже існуючим IT-комплексом може бути утруднена.

Ідея «ЦОД з коробки» або «блокового ЦОД», звичайно, залучає ті, хто збирається будувати дати-центри. Наприклад, ми зараз зайняті підготовкою проектів по будівництву біля десяти ЦОД у різних регіонах країни. Головне, не помилитися в прогнозі потужностей. У цій ситуації дуже привабливої виглядає можливість покупки готового блоку, до якого можна додати другий або третій. Ми уважно стежимо за тенденціями розвитку подібних рішень і іноді їх вибираємо. Однак ціна часом занадто висока, тому в ряді випадків вигідніше просто встановити в приміщенні стійки й розмістити в них устаткування».

Варто розрізняти програмно-апаратні платформи великих вендорів і «хмара з коробки» – по суті незалежне від апаратних засобів інтегроване програмне рішення, що складається з великої кількості функціональних модулів. З ростом затребуваності й здешевленням послуг IaaS і SaaS, надаваних зовнішніми провайдерами, будівництво власного ЦОД і частки хмари може бути актуально тільки для великих підприємств. Використовувати готові рішення або власні програмні розробки на базі референсних архітектур і платформ із відкритим кодом, наприклад OpenStack, – питання IT-стратегії. Як показує практика, коли закінчується перехідний період прийняття технологій ринком, готові промислові рішення виявляються дешевше власних розробок.

Вимоги замовників до побудови приватної хмари розділяють на дві групи: до першого ставиться все, що пов'язане зі зручністю й доступністю розташовуваних у хмарі корпоративних сервісів, до другого – безпека й захист даних у хмарі від несанкціонованого доступу. Особлива увага приділяється масштабованості рішень, простоті й гнучкості їхньої інтеграції з наявної IT-інфраструктурою.

Сьогодні замовники краще інформовані про можливості хмар, тому їхнього очікування від реалізації таких проектів стають усе більше конкретними. Шлях до приватної хмари рекомендуємо починати з аудита сформованої інфраструктури, у процесі якого з'ясовується, яка частка застарілих додатків, у чому складаються актуальні вимоги замовника до продуктивності інфраструктури, якій специфіці бізнесу і яким зовнішнім нормативним вимогам повинне відповідати створювана приватна хмара.

До числа важливих тенденцій фахівці цієї компанії відносять зміну моделі надання корпоративних хмарних сервісів: сьогодні підприємства зацікавлені в одержанні готових рішень із хмари, без додаткових зусиль по їхньому об'єднанню в рамках корпоративної мережі.

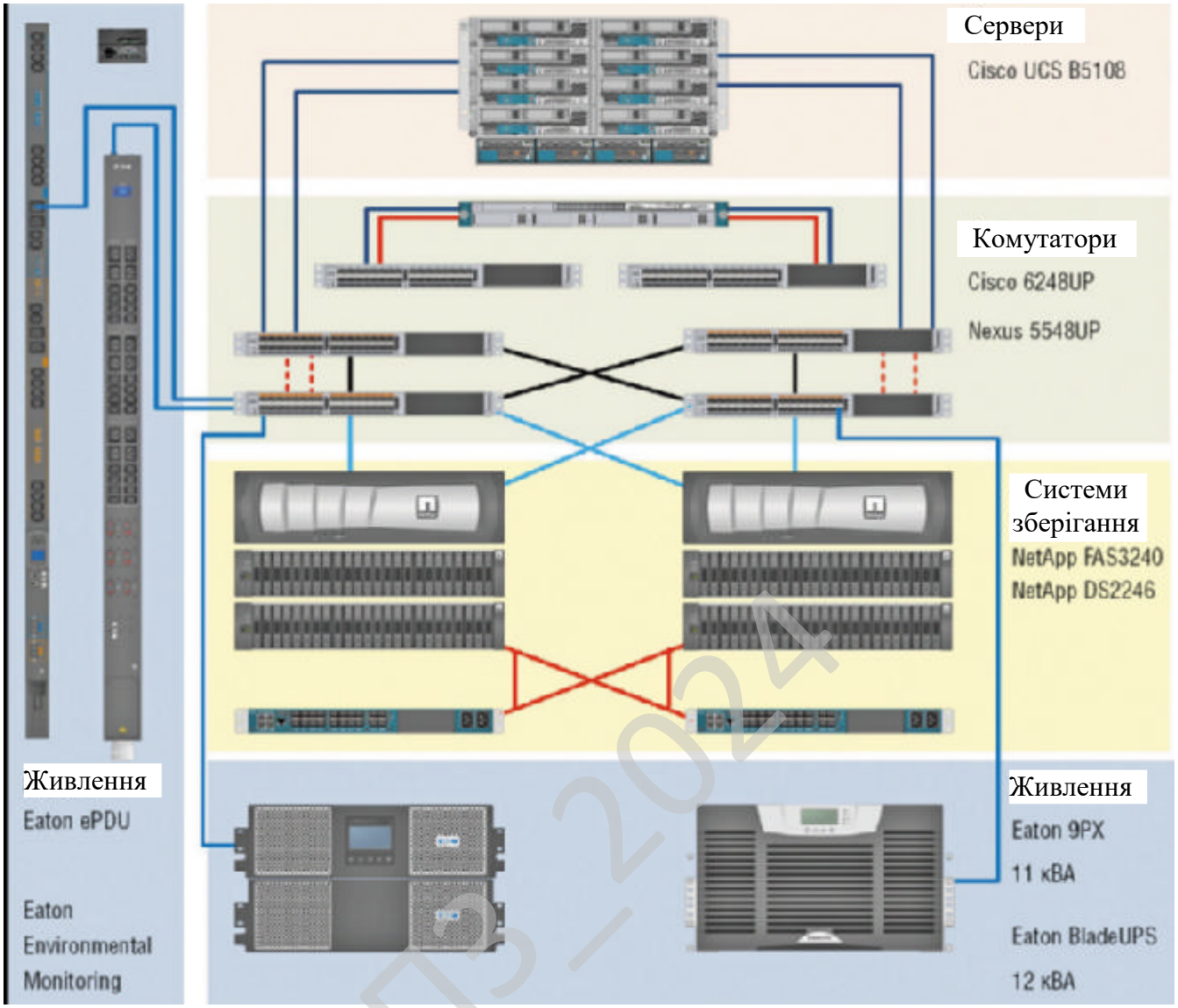

Рисунок 3.1 – Структурна схема системи

Недарма все частіше експерти говорять про швидку появу єдиної сумісної (конвергентної) програмного, комунікаційного й апаратного середовища, у якій будуть уніфіковані як самі IT-сервіси, так і моделі їхнього надання. По суті, це конструктор, на основі якого можна буде створювати комплексні рішення, що максимально враховують специфіку бізнесу, галузі й вимоги законодавства, що особливо важливо для компаній з банківського сектора, для державних структур і організацій охорони здоров'я.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно. Функціональна схема складається з двох великих блоків:

– Блок визначення об’єктів моніторингу інтегрованих систем для приватної хмари.

– Блок визначення функцій моніторингу інтегрованих систем для приватної хмари.

З рисунку видно, що блок визначення об’єктів моніторингу інтегрованих систем для приватної хмари локальної мережі складається з трьох блоків:

– Моніторинг трафіку інтегрованих систем для приватної хмари.

– Моніторинг обладнання інтегрованих систем для приватної хмари.

– Моніторинг ресурсів інтегрованих систем для приватної хмари.

Моніторинг трафіку інтегрованих систем для приватної хмари використовується для контролю вхідного та вихідного трафіку інтегрованих систем для приватної хмари. Він включає у себе контроль підключених інтерфейсів, статистику подій по основним мережним протоколам: TCP, UDP, IP та ICMP.

TCP – один з основних мережних протоколів Інтернету, призначений для управління передачею даних в мережах і підмережах TCP/IP.

UDP – один із протоколів в стеку TCP/IP. Від протоколу TCP він відрізняється тим, що працює без встановлення з'єднання. UDP – це один з найпростіших протоколів транспортного рівня моделі OSI, котрий виконує обмін дейтаграмами без підтвердження та гарантії доставки.

IP – найбільш широко розповсюджена реалізація ієрархічної схеми мережної адресації. Використовуваний в мережі Інтернет, протокол відповідає за адресацію пакетів, але не відповідає за встановлення з'єднань, не є надійним і дозволяє реалізувати тільки негарантовану доставку даних.

Рисунок 3.2 – Функціональна схема системи

ICMP – мережний протокол, що входить в стек протоколів TCP/IP. В основному ICMP використовується для передачі повідомлень про помилки й інші виняткові ситуації, що виникли при передачі даних. Також на ICMP покладають деякі сервісні функції, зокрема на основі цього протоколу заснована дія таких загальновідомих утиліт як ping та traceroute.

Моніторинг обладнання інтегрованих систем для приватної хмари включає в себе побудову списку наявного обладнання інтегрованих систем для приватної хмари та здійснення його контролю. До мережного обладнання інтегрованих систем для приватної хмари, що підлягає моніторингу інтегрованих систем для приватної хмари, відносяться: персональні комп'ютери, ноутбуки, сервери, принтери, IP-телефони.

Моніторинг ресурсів інтегрованих систем для приватної хмари дозволяє переглядати та завантажувати наявні в мережі ресурси, а також розміщувати чи приховувати для загального доступу свої ресурси. До ресурсів інтегрованих систем для приватної хмари локальної мережі відносяться: файли, мультимедіа, бази даних, сервіси інформаційної безпеки, список користувачів.

Блок визначення функцій моніторингу інтегрованих систем для приватної хмари складається з наступних блоків:

- Робота з ресурсами мережі.
- Робота з сесіями.
- Моніторинг трафіку інтегрованих систем для приватної хмари.
- Робота з файлами.
- Монітор з'єднань.
- Статистика подій.
- Функції для роботи з мережею.

Розглянемо детальніше кожний з блоків.

Робота з ресурсами мережі включає в себе:

– Визначення доступних ресурсів інтегрованих систем для приватної хмари.

– Закриття локального ресурсу.

– Відкриття локального ресурсу.

– Приховання й показ ресурсів інтегрованих систем для приватної хмари.

Система відображає наявні у мережі ресурси у вигляді дерева. Пошук ресурсів інтегрованих систем для приватної хмари можна здійснювати по заданим умовам: локальні чи глобальні ресурси; всі ресурси, тільки файли, чи тільки принтери, тощо.

Можна додавати до загальних ресурсів інтегрованих систем для приватної хмари мережі свої власні, а також закривати їх потім.

Робота з сесіями включає в себе:

– Одержання списку поточних сесій.

– Завершення сесій.

Програма дозволяє переглянути список відкритих сесій, що включає в себе: назву сесії, користувача, що її розпочав, номер сесії, час роботи та час очікування.

Моніторинг трафіку інтегрованих систем для приватної хмари включає в себе:

– Визначення підключених інтерфейсів.

– Визначення вхідного та вихідного трафіку інтегрованих систем для приватної хмари.

Розроблена система відображає всі інтерфейси приєднані до комп'ютера, на якому запущена програма, їх MAC-адреси та вхідний і вихідний трафік на кожному з них.

Робота з файлами включає в себе:

– Одержання списку відкритих файлів.

– Закриття відкритого файлу.

Можна переглянути, які з Ваших файлів, що Ви відкрили для загального доступу, переглядають по мережі. Програма відобразить список файлів та користувачів, які їх переглядають. Також можна відкрити чи закрити файл.

Монітор з'єднань включає в себе:

– Відстеження TCP– з'єднань.

– Відстеження UDP– з'єднань.

Система фіксує всі підключення по TCP– та UDP-протоколу, та виводить їх на екран у форматі:

IP-адреса : порт_призначення.

Статистика подій включає в себе відстеження подій в наступних протоколах:

– TCP-протокол.

– UDP-протокол.

– IP-протокол.

– ICMP-протокол.

Статистика ведеться по цілому ряду параметрів. Наприклад для TCP-протоколу фіксується: Тип алгоритму повторної передачі, мінімальний тайм-аут, максимальний тайм-аут, максимальна кількість помилок з'єднання, активні з'єднання, пасивні з'єднання, невдалі спроби відкриття, скидання встановлених з'єднань, отримані сегменти, надіслані сегменти, повторно передані сегменти, помилки тощо.

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання дипломного проектування, наведена на рисунку 3.3. Першим процесом, який завантажується у мережі є процес виведення головного вікна програми. Він взаємодіє з наступними процесами:

– Процес моніторингу інтегрованих систем для приватної хмари трафіку інтегрованих систем для приватної хмари.

– Процес відстеження пакетів даних.

– Процес моніторингу інтегрованих систем для приватної хмари відкритих сесій.

– Процес моніторингу інтегрованих систем для приватної хмари ресурсів інтегрованих систем для приватної хмари локальної мережі.

Рисунок 3.3 – Діаграма взаємодії процесів

Процес моніторингу інтегрованих систем для приватної хмари трафіку інтегрованих систем для приватної хмари взаємодіє з наступними процесами:

– Процес моніторингу інтегрованих систем для приватної хмари вхідного трафіку інтегрованих систем для приватної хмари.

– Процес моніторингу інтегрованих систем для приватної хмари вихідного трафіку інтегрованих систем для приватної хмари.

– Процес моніторингу інтегрованих систем для приватної хмари підключених інтерфейсів.

Процес відстеження пакетів даних взаємодіє з наступними процесами:

– Процес відстеження пакетів UDP-протоколу.

– Процес відстеження пакетів IP-протоколу.

– Процес відстеження пакетів ICMP-протоколу.

Процес моніторингу інтегрованих систем для приватної хмари відкритих сесій взаємодіє з наступними процесами:

– Процес перегляду сесій.

– Процес закриття сесій.

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього здійснюється пошук ресурсів інтегрованих систем для приватної хмари локальної мережі. На основі знайдених ресурсів інтегрованих систем для приватної хмари локальної мережі відбувається дерева цих ресурсів інтегрованих систем для приватної хмари. Для користувача на екран виводяться:

– Дерево ресурсів інтегрованих систем для приватної хмари мережі.

– Список відкритих по мережі файлів.

– Список відкритих сесій.

На основі цих даних користувач може здійснити один із наступних виборів:

– Відкрити ресурс.

– Закрити ресурс.

Програма здійснює моніторинг трафіку інтегрованих систем для приватної хмари, на основі чого виконується виведення на екран:

– Вхідного трафіку інтегрованих систем для приватної хмари.

– Вихідного трафіку інтегрованих систем для приватної хмари.

– Виведення статистики.

Також виконується відстеження підключених інтерфейсів.

При відстежуванні пакетів даних здійснюється слідкування за пакетами ICMP-, IP- та UDP-протоколів.

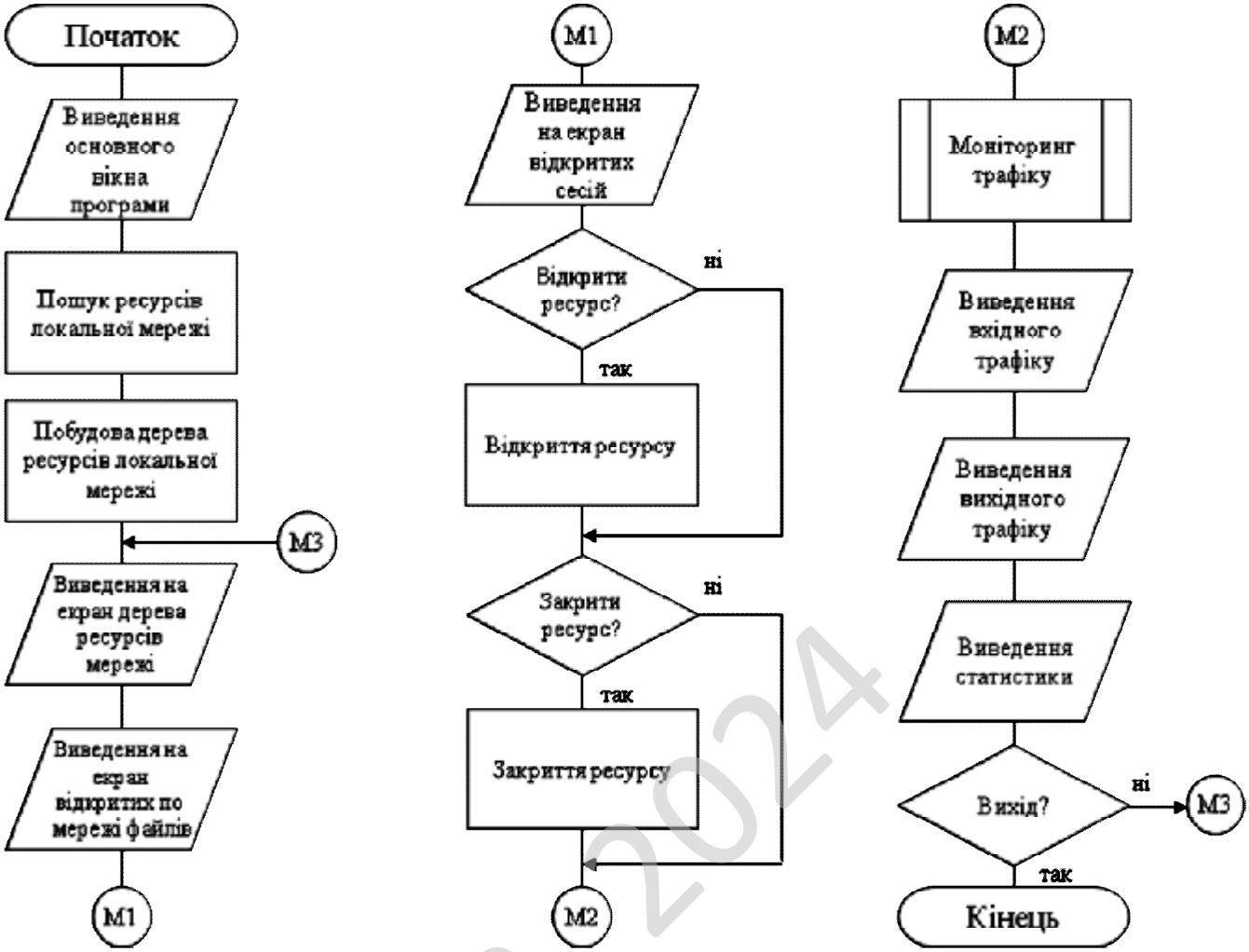

Рисунок 4.1 – Блок-схема основної програми

На рисунку 4.2 представлено блок-схему підпрограми відкриття мережного ресурсу. Для додавання загального ресурсу виконуються наступні операції:

– Введення користувачем шляху до ресурсу.

– Введення імені ресурсу.

– Встановлення прав доступу до ресурсу.

– Встановлення максимальної кількості підключень до ресурсу.

– Додавання ресурсу.

– Якщо ресурс потрібно захистити паролем, вводяться паролі для доступу до запису/читання та для доступу тільки до читання.

– Встановлюються паролі доступу до ресурсу.

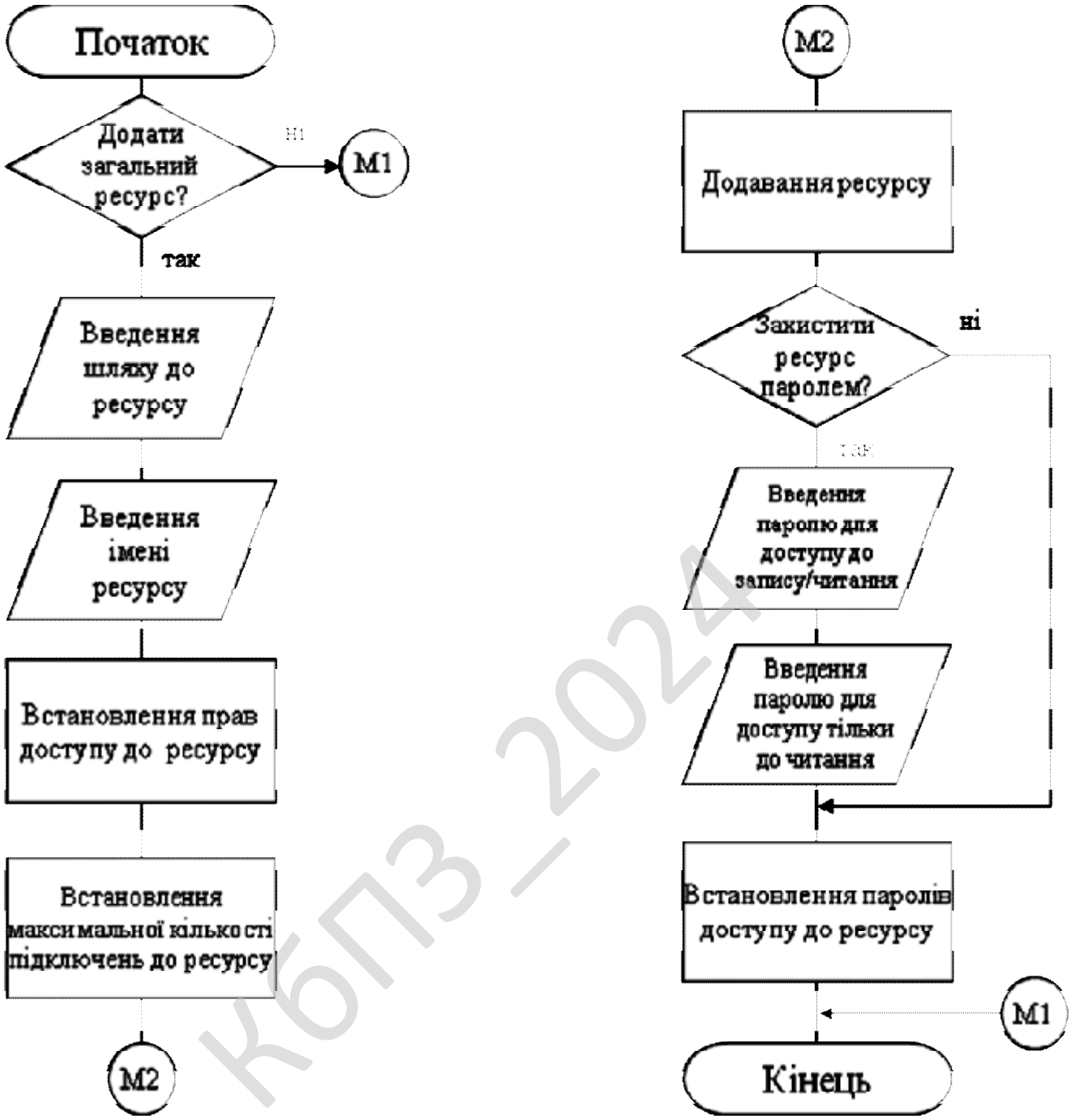

Рисунок 4.2 – Блок-схема підпрограми відкриття мережного ресурсу

Для створення системи інтеграції для приватної хмари на Python спроектовано архітектуру, що включає кілька модулів.

Система складається з основних компонентів:

1. Модуль аутентифікації і авторизації забезпечує безпечний доступ до ресурсів хмари.

2. Модуль управління даними відповідає за зберігання та обробку даних у хмарному середовищі.

3. Модуль Обчислень виконує розрахунки та обробку запитів користувачів.

4. Модуль моніторингу та логування фіксує всі дії в хмарі для забезпечення прозорості та діагностики роботи системи.

У коді представлені базові функції для кожного з модулів з прикладом того, як інтеграція може функціонувати.

```
import hashlib
import logging
from datetime import datetime

# Налаштування для логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

class AuthModule:
    def __init__(self):
        self.users = {}

    def register_user(self, username, password):
        hashed_password = hashlib.sha256(password.encode()).hexdigest()
        self.users[username] = hashed_password
        logging.info('User registered successfully')

    def authenticate_user(self, username, password):
        hashed_password = hashlib.sha256(password.encode()).hexdigest()
        if self.users.get(username) == hashed_password:
            logging.info('User authenticated successfully')
            return True
        else:
            logging.warning('Authentication failed')
            return False

class DataModule:
    def __init__(self):
        self.data_storage = {}

    def store_data(self, key, value):
```

```
            self.data_storage[key] = value
            logging.info('Data stored successfully')

        def retrieve_data(self, key):
            data = self.data_storage.get(key, 'Data not found')
            logging.info('Data retrieved successfully')
            return data

    class ComputeModule:
        def compute_sum(self, numbers):
            result = sum(numbers)
            logging.info('Sum computed successfully')
            return result

        def compute_average(self, numbers):
            if numbers:
                result = sum(numbers) / len(numbers)
                logging.info('Average computed successfully')
                return result
            else:
                logging.warning('Empty list provided for average computation')
                return 0

    class MonitoringModule:
        def __init__(self):
            self.logs = []

        def log_event(self, event):
            timestamp = datetime.now().isoformat()
            self.logs.append(f'{timestamp} - {event}')
            logging.info('Event logged successfully')

        def get_logs(self):
            return self.logs

    class CloudIntegrationSystem:
        def __init__(self):
            self.auth_module = AuthModule()
            self.data_module = DataModule()
            self.compute_module = ComputeModule()
            self.monitoring_module = MonitoringModule()
```

```
        def perform_operations(self):
            self.auth_module.register_user('user1', 'password123')
            auth_result = self.auth_module.authenticate_user('user1',
'password123')

            if auth_result:
                self.data_module.store_data('data_key', 'sample_data')
                retrieved_data = self.data_module.retrieve_data('data_key')

                numbers = [10, 20, 30, 40]
                sum_result = self.compute_module.compute_sum(numbers)
                average_result = self.compute_module.compute_average(numbers)

                self.monitoring_module.log_event('Operations performed')
                logs = self.monitoring_module.get_logs()

                return {
                    'auth_result': auth_result,
                    'retrieved_data': retrieved_data,
                    'sum': sum_result,
                    'average': average_result,
                    'logs': logs
                }

    # Запуск системи інтеграції приватної хмари
    cloud_system = CloudIntegrationSystem()
    result = cloud_system.perform_operations()
    print(result)
```

Опис архітектури і модулів:

1. AuthModule забезпечує реєстрацію та аутентифікацію користувачів використовуючи хешування паролів.

2. DataModule забезпечує зберігання даних у пам'яті та надає можливість швидкого доступу до них.

3. ComputeModule дозволяє виконувати прості обчислення такі як сума і середнє значення.

4. MonitoringModule веде журнал всіх подій для забезпечення діагностики та моніторингу.

Результати обчислень і перевірки коректності. Для підтвердження

правильності роботи наданої системи перевіряється результат кожної з операцій у модулі CloudIntegrationSystem:

– Система успішно реєструє та аутентифікує користувача.

– Дані зберігаються і повертаються коректно.

– Розрахунок суми та середнього значення проходить без помилок.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму UMAC (код автентифікації повідомлення на основі універсального гешування) – один з видів коду автентичності повідомлень (MAC).

Швидка «універсальна» функція використовується, для того, щоб гешувати вхідне повідомлення M у короткий рядок. До цього рядка потім застосовується функція XOR із псевдовипадковим значенням, у результаті чого ми одержуємо тег UMAC:

$$Tag=H_{K1}(M)\oplus F_{K2}(Nonce)$$

де K1 і K2 – секретні випадкові ключі, які мають одержувач і відправник.

Звідси видне, що безпека UMAC залежить від того, яким випадковим способом відправник і одержувач вибрали таємну геш-функцію й псевдовипадкову послідовність. При цьому значення Nonce міняється кожний такт. Через використання Nonce, приймач і передавач повинні знати час відправлення повідомлення й принцип створення значення Nonce. Замість цього можна використовувати в якості Nonce будь-яке інше неповторюване значення, наприклад порядковий номер повідомлення. При цьому даний номер не зобов'язано бути секретним, головне щоб він не повторювався.

UMAC розрахований на використання 32-х, 64-х, 92-х, і 128-бітових тегів, залежно від необхідного рівня безпеки. UMAC звичайно використовується разом з алгоритмом шифрування AES.

Функція створення ключа й псевдовипадкової послідовності

Створення псевдовипадкових байтів необхідно для роботи UHASH і при створенні тегів

Вибір блокового шифру

Для своєї роботи UMAC використовує блоковий шифр, вибір якого визначають наступні константи:

– BLOCLEN – довжина, у байтах, блоку з яким працює блоковий шифр.

– KEYLEN – довжина, у байтах, ключа блокового шифру.

При цьому використовується функція

– ENCIPHER(K,P) – зашифрувати рядок P з BLOCLEN байтів, використовуючи ключ K.

Приклад: якщо використовується AES з 16-байтним ключем, то BLOCLEN буде рівним 16(тому що AES працює з 16-байтними блоками).

KDF – функція створення ключа

Ця функція генерує послідовність псевдовипадкових байтів, використовуваних для ключових геш-функцій.

Вхід:

– K – рядок довжиною KEYLEN байт. / / Ключ блокового шифру.

– Index – ненегативне ціле число менше, чим 2 ^ 64.

– Numbytes – ненегативне ціле число менше, чим 2 ^ 64.

Вихід:

– Y – рядок довжини numbytes байт.

PDF: функція створення псевдовипадкового числа

Ця функція ухвалює ключ і даний час і повертає псевдовипадкове число для використання його в тегу покоління. За допомогою цієї функції можуть бути отримані числа довжиною 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– Nonce – рядок довжиною від 1 до BLOCKLEN байт.

– Taglen – ціле число 4, 8, 12 або 16.

Вихід:

– Y – послідовність байтів довжини taglen.

Генерація UMAC-тегів

Генерація UMAC-тегів відбувається за допомогою UHASH функції при використанні Nonce значенні й отриманої до цього рядка. Їхня довжина може бути 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– M – рядок довжиною менше 267 біт.

– Nonce – випадкове число від 1 до BLOCKLEN байт.

– Taglen – ціле 4, 8, 12 або 16.

Вихід:

– Тег, послідовність байтів довжиною taglen.

Алгоритм обчислення тегів:

Hashedmassage = UHASH(K, M, Taglen)

Pad = PDF(K, nonce, Taglen)

Tag = Pad xor Hashedmassage

UMAC-32 UMAC-64 UMAC-96 UMAC-128

Дані позначення містять у своїй назві певне значення довжини тегу:

– UMAC-32 (K, M, Nonce) = UMAC (K, M, Nonce, 4).

– UMAC-64 (K, M, Nonce) = UMAC (K, M, Nonce, 8).

– UMAC-96 (K, M, Nonce) = UMAC (K, M, Nonce, 12).

– UMAC-128 (K, M, Nonce) = UMAC (K, M, Nonce, 16).

Універсальна функція гешування(UHASH)

UHASH – універсальна функція гешування, серцевина алгоритму UMAC. UHASH – функція працює в три етапи. Спочатку до вхідного повідомлення застосовується L1-HASH, потім до цього результату застосовується L2-HASH і, нарешті, до результату застосовується L3-HASH . Якщо при цьому довжина

вхідного повідомлення не більш 1024 біт, то L2-HASH не використовується. Тому що функція L3-hash повертає тільки слово довжини 4 байта, те якщо потрібно одержати геш довжини більше 4 байт, здійснюється кілька ітерацій даної трирівневої схеми.

Універсальна функція

Нехай функція гешування вибирається із класу геш-функцій H, які відображають повідомлення в D, набір усіляких образів повідомлення. Цей клас називається універсальним, якщо для яких-небудь окремих пар повідомлень, існує на безлічі H/D функцій, функція, яка відображає їх в елемент D. Зміст цієї функції в тому, що якщо третя сторона прагне замінити одне повідомлення іншим, але при цьому вважає, що геш-функція була обрана абсолютно випадково, те ймовірність не виявлення підміни стороною, що ухвалює, прагне до 1/D.

L1-hash – перший етап

L1-hash розбиває повідомлення на шматки з 1024 байт і до кожного шматка застосовує алгоритм гешування називаний NH. Вихідний результат алгоритму NH в 128 раз менше вхідного.

L2-hash – другий етап

L2-hash працює з виходом L1-hash, використовує поліноміальний алгоритм POLY. Другий етап гешування використовується, тільки якщо довжина вхідного повідомлення більше 16 мегабайт. Використання алгоритму POLY потрібно для того, щоб уникнути тимчасову атаку. На виході з алгоритму POLY виходить 16 байтне число.

L3-hash – третій етап

Цей етап потрібно для того щоб з вихідних 16 байтів алгоритму L2-hash одержати 4-байтне значення.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Загальний вигляд інтерфейсу програми подається на рисунку 5.1, на якому зображено головне вікно програми.

Для перегляду інформації про ресурси мережі використовуються наступні області:

– Дерево мережних ресурсів інтегрованих систем для приватної хмари.

– Відкриті по мережі файли.

– Відкриті сесії.

– Ресурси.

Ці області дублюються пунктами головного меню, в яке крім того входять такі пункти:

– Параметри.

– Довідка.

Відображення у дереві мережних ресурсів інтегрованих систем для приватної хмари можна змінювати, вибираючи розміщення ресурсів інтегрованих систем для приватної хмари (Локальні, Глобальні, Обрані), тип ресурсів інтегрованих систем для приватної хмари (Всі, Диски, Принтери), а також використання ресурсів інтегрованих систем для приватної хмари (Все, З'єднання, Перегляд).

Під вікном мережних ресурсів інтегрованих систем для приватної хмари розташовані клавіші для запуску TCP/IP монітора та виведення статистики.

При перегляді відкритих по мережі файлів, відображається ID файлу, його повна адреса та дані про користувача, який відкрив даний файл.

За допомогою програмних кнопок можливо віддалено закрити обраний файл або знову його відкрити.

Аналогічно можна керувати відкритими сесіями.

Для кожної сесії вказується наступна інформація:

– Назва сесії.

– Користувач, який створив сесію.

– Номер.

– Час.

– Час очікування.

Для закриття та поновлення сесій використовуються кнопки Закрити сесію та Відкрити сесію.

На рисунку 5.2 зображено вікно довідки про програму.

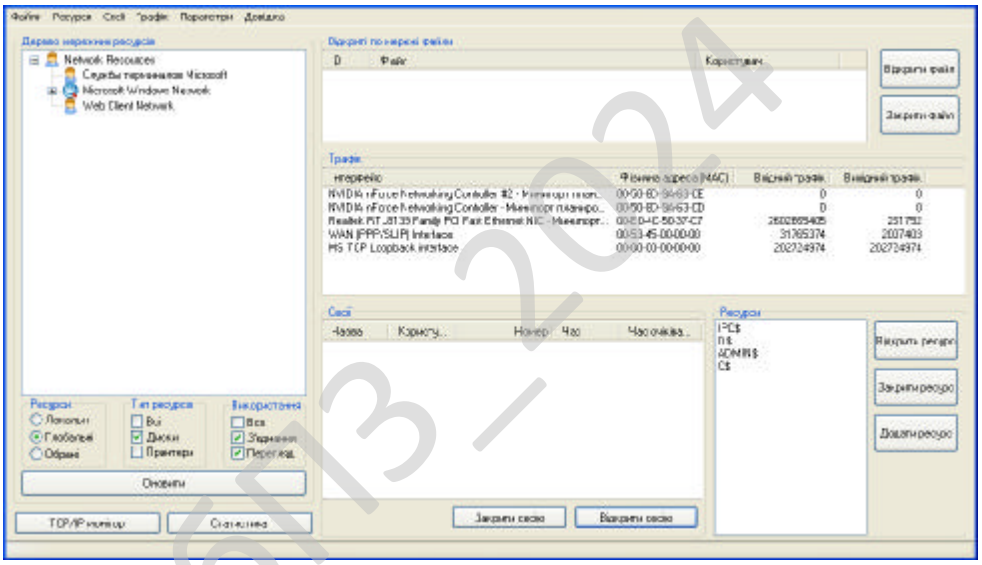

Рисунок 5.1 – Головне вікно програми

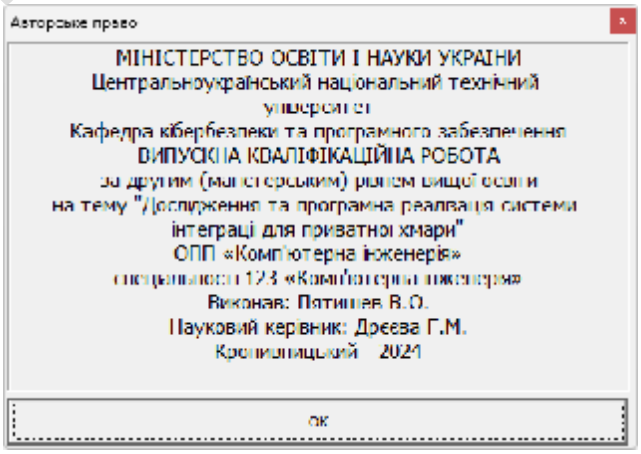

Рисунок 5.2 – Вікно довідки

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи інтеграції для приватної хмари.

Метою розробки є дослідження та програмна реалізація системи інтеграції для приватної хмари.

Об'єктом дослідження є процес інтеграції для приватної хмари.

Предметом дослідження є методи інтеграції для приватної хмари.

Методи дослідження базуються на методах теорії комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод інтеграції для приватної хмари.

– Розроблено вітчизняний продукт інтеграції для приватної хмари, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмна реалізація системи інтеграції для приватної хмари можуть бути цікавими для таких категорій користувачів та організацій(рисунок 7.1).

Компанії та корпорації, що використовують приватні хмарні середовища — організації, які прагнуть оптимізувати та автоматизувати роботу своїх приватних хмарних рішень, зможуть зацікавитися новими підходами до інтеграції системи. Це може підвищити ефективність управління ресурсами та забезпечити кращу безпеку даних.

Розробники програмного забезпечення для хмарних платформ — IT-фахівці та розробники, що спеціалізуються на створенні, інтеграції та підтримці хмарних сервісів, можуть знайти корисними технічні інсайти та рішення, що оптимізують процеси взаємодії між різними компонентами хмари.

Інтегратори систем — компанії, які спеціалізуються на інтеграції технологічних рішень, можуть використовувати дане дослідження для підвищення якості своїх послуг, адаптуючи нові методики інтеграції та впровадження.

Організації з високими вимогами до безпеки — приватні хмарні рішення часто використовуються в секторах, де безпека є критично важливою, наприклад у банківській сфері, фінансових організаціях, медичних установах. Ці організації можуть скористатися новими рішеннями, що підвищують рівень захищеності даних.

Дослідники та академічна спільнота — аспіранти, студенти, науковці в галузі хмарних технологій, віртуалізації та безпеки можуть використовувати результати дослідження для подальших робіт та розробок.

Постачальники рішень для приватних хмар — компанії, що надають послуги хмарної інфраструктури на базі приватних хмар, можуть використовувати рішення для вдосконалення своїх пропозицій клієнтам.

Рисунок 7.1 – Цільова аудиторія

Цільова аудиторія залежить від специфіки реалізації і може бути розширена відповідно до потреб унікальних функцій та особливостей даного проекту.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості програмної реалізації системи інтеграції для приватної хмари можна застосувати метод експертних оцінок, який передбачає залучення фахівців для визначення ключових критеріїв і присвоєння їм вагових коефіцієнтів.

Вибираються найважливіші критерії, які будуть оцінювати експерти (рисунок 7.2).

Рисунок 7.2 – Критерії оцінки

Кожному критерію присвоюється ваговий коефіцієнт (за сумою ваг отримуємо 1 або 100%), що відображає його важливість для привабливості системи: функціональність (f): 0,3, масштабованість (s): 0,2, безпека (b): 0,25, економічна ефективність (e): 0,15, зручність використання (u): 0,1.

Кожен експерт (IT-архітектор, розробник, спеціаліст з безпеки) оцінює програмну реалізацію за шкалою від 1 до 10 (де 10 – найвищий бал) для кожного критерію. Зведені результати вносимо в таблицю 7.1.

Таблиця 7.1 – Зведені результати експертних оцінок

Критерій	Експерт 1	Експерт 2	Експерт 3	Середнє значення
Функціональність (F)	9	8	9	8,67
Масштабованість (S)	7	8	8	7,67
Безпека (B)	8	9	9	8,67
Економічна ефективність (E)	6	7	7	6,67
Зручність використання (U)	8	7	7	7,33

Для кожного критерію середнє значення оцінки множиться на ваговий коефіцієнт. Сума результатів дає загальну інтегральну оцінку привабливості системи.

Загальна оцінка = (F×8,67) + (S×7,67) + (B×8,67) + (E×6,67) + (U×7,33) = (0,3×8,67) + (0,2×7,67) + (0,25×8,67) + (0,15×6,67) + (0,1×7,33) = 2,60 + 1,53 + 2,17 + 1,00 + 0,73 = 8,03.

Одержана інтегральна оцінка привабливості системи – 8,03 із 10 можливих, що свідчить про високу привабливість реалізації системи для інтеграції приватної хмари.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації системи інтеграції для приватної хмари пропонуємо використовувати метод COCOMO II. Цей метод корисний для оцінки вартості великих проектів, де враховуються обсяги коду, складність системи та специфіка команди. COCOMO II використовує емпіричні

дані для прогнозування зусиль, часу та витрат. Основні етапи: визначення обсягів роботи в кількості рядків коду, використання параметрів, таких як досвід команди, складність вимог, вимоги до надійності, розрахунок зусиль, часу і вартості, використовуючи формули COCOMO II.

Для проекту з програмної реалізації інтеграції для приватної хмари доцільно використовувати COCOMO II адже саме він підходять для IT-проектів зі складною архітектурою та специфічними функціональними вимогами.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Впровадження системи інтеграції для приватної хмари може призвести до значної економічної ефективності для організації. Варіанти економії подано на рисунку 7.3.

Впровадження системи інтеграції за рахунок сукупного зниження витрат і підвищення продуктивності принесе економію підприємству та підвищення ефективності роботи.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Алгоритм просування проєкту програмної реалізації системи інтеграції для приватної хмари може включати кроки, представлені на рисунку 7.4.

Цей алгоритм дозволяє комплексно та послідовно реалізувати стратегію просування проєкту системи інтеграції для приватної хмари, забезпечуючи вихід на ринок і побудову стійких відносин з клієнтами та партнерами.

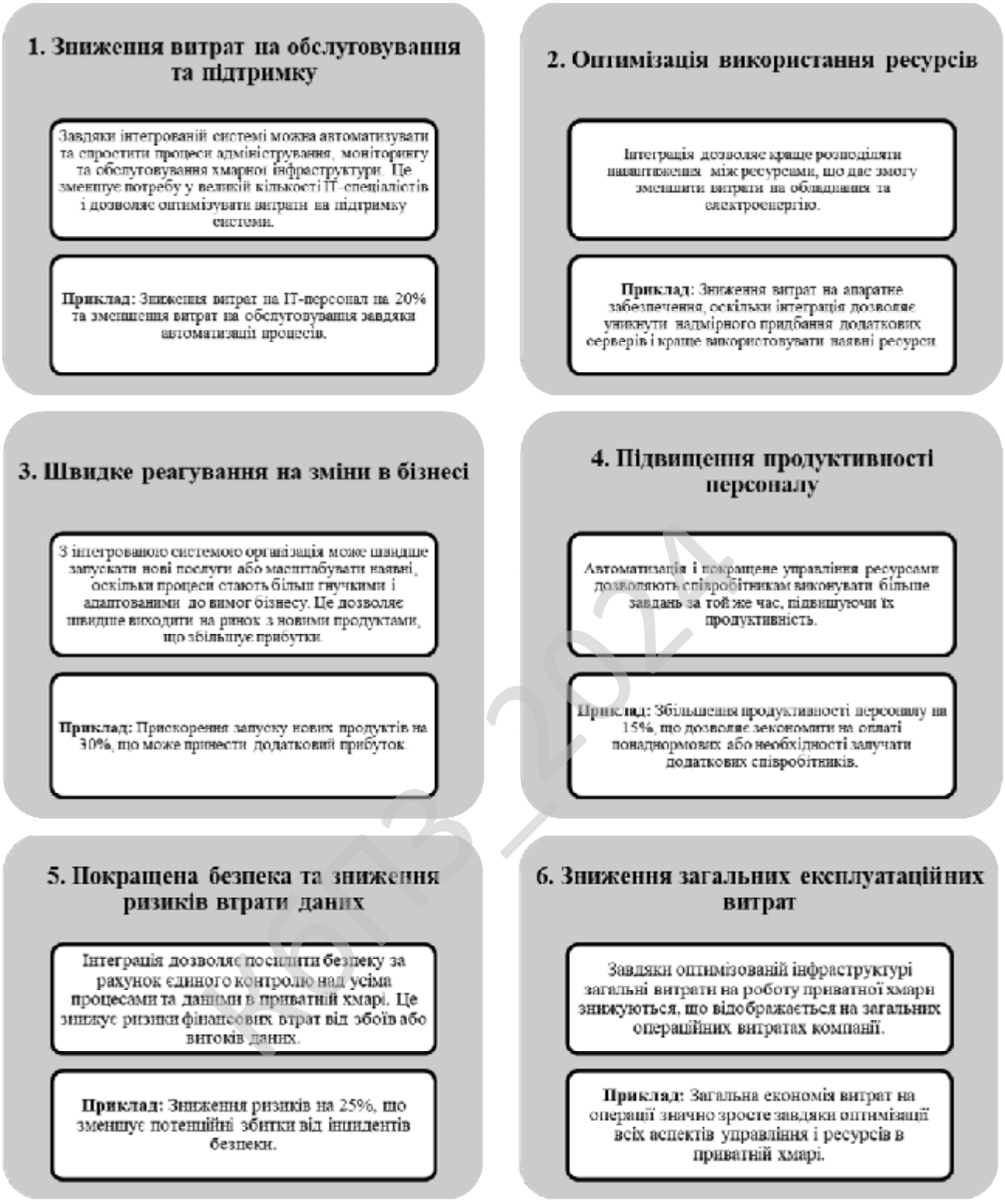

Рисунок 7.3 – Економічна ефективність від реалізації проєкту для клієнта

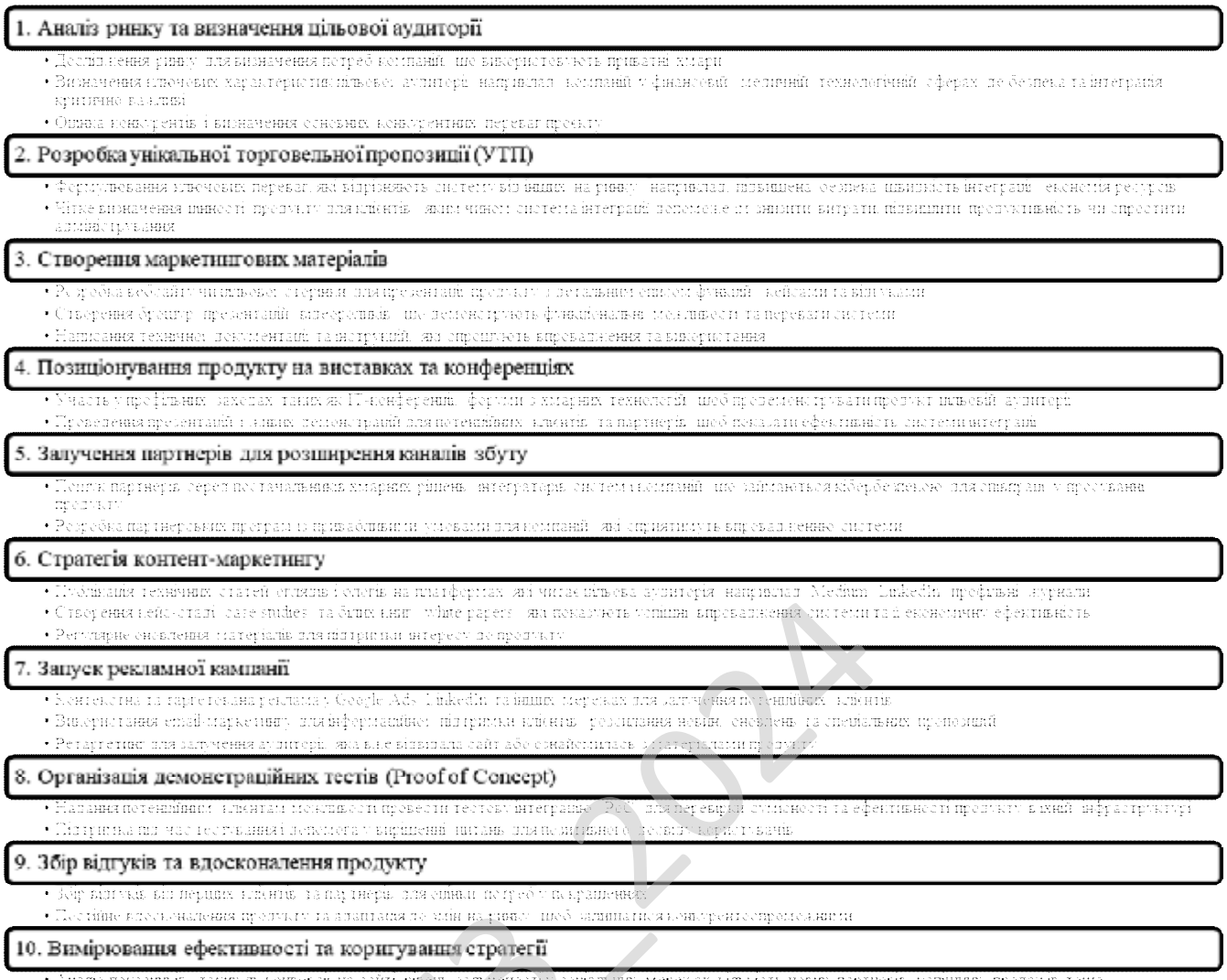

Рисунок 7.4 – Алгоритм просування проєкту

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту та шляхів реалізації проєкту програмної реалізації системи інтеграції для приватної хмари можна запропонувати наступні стратегії (рисунок 7.4).

1. Аналіз та вибір ефективних каналів збуту

- **Прямий продаж через власну команду продажів:** Залучення фахівців з B2B-продажів для роботи з великими компаніями, що потребують інтеграції приватних хмарних систем. Прямий продаж особливо підходить для залучення великих корпоративних клієнтів, де потрібна детальна персоналізація рішення.
- **Партнерські канали:** Створення партнерських відносин із системними інтеграторами, провайдерами хмарних послуг та компаніями, що займаються безпекою. Це дозволить забезпечити вихід на ширшу аудиторію та збільшити обсяги збуту.
- **Інтернет-канали та електронна комерція:** Пропозиція ліцензій або демоверсій через сайт компанії, з можливістю оформлення заявки на тестування продукту. Це забезпечить залучення середніх і малих підприємств, які віддають перевагу самостійному ознайомленню з продуктом.

2. Оптимізація каналів збуту

- **Сегментація цільових клієнтів:** Виділення різних сегментів (великі корпорації, середні підприємства, державні установи) і адаптація підходів до кожного сегмента. Наприклад, для великих підприємств акцент на комплексні рішення та спеціалізовану технічну підтримку, а для малих компаній – готові до використання пакети з простою інтеграцією.
- **Розробка програми лояльності для партнерів:** Введення бонусів, знижок або пільгових умов для партнерів, що успішно реалізують продукт. Це може стимулювати більше партнерів активно просувати систему інтеграції.
- **Аутсорсинг продажів через зовнішні агентства:** Це може дозволити охопити ринки або сегменти, де компанія не має власних представництв. Агентства можуть займатися первинними продажами, а внутрішня команда – супроводом клієнтів та налаштуванням системи.

3. Оптимізація процесів залучення та утримання клієнтів

- **Надання демоверсій або безкоштовного тестування:** Можливість спробувати систему протягом обмеженого періоду може підвищити рівень довіри до продукту та допоможе клієнтам зрозуміти його цінність.
- **Регулярна комунікація та зворотний зв'язок:** Забезпечення чіткого каналу комунікації для обговорення функцій, вирішення проблем та адаптації до вимог клієнта, що дозволить утримувати клієнтів у довгостроковій перспективі.
- **Реферальні програми:** Запровадження програм, де клієнти можуть рекомендувати продукт іншим і отримувати бонуси або знижки. Це допоможе залучати нових клієнтів без значних витрат на маркетинг.

Рисунок 7.5 – Оптимізація каналів збуту та шляхів реалізації

Ці кроки допоможуть оптимізувати канали збуту та шляхи реалізації проєкту програмної реалізації системи інтеграції для приватної хмари, забезпечуючи максимальний охоплення цільової аудиторії, підвищення конкурентоспроможності продукту на ринку та зниження витрат на його просування.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовими факторами успіху проєкту програмної реалізації системи інтеграції для приватної хмари є ряд процедур та переваг (рисунок 7.6).

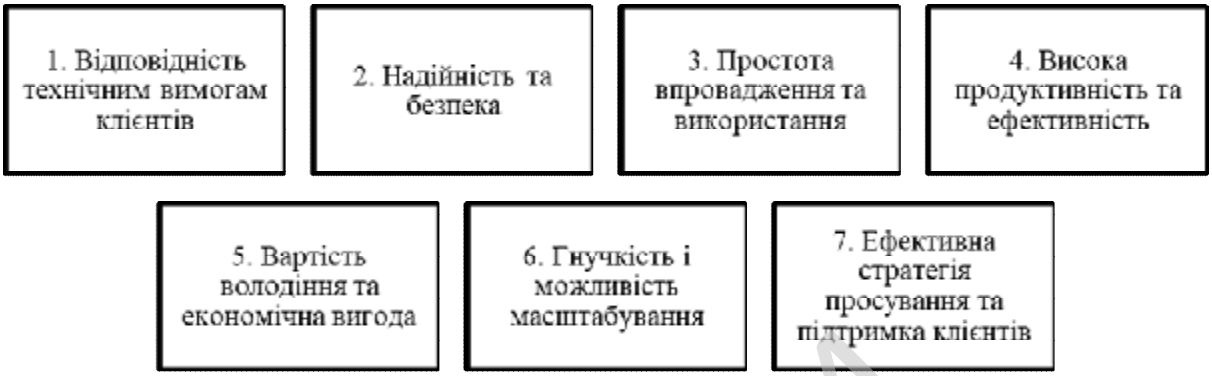

Рисунок 7.6 – Ключові фактори успіху проєкту

Ці фактори успіху дозволять проєкту не лише задовольнити потреби цільової аудиторії, але й виділитися на ринку як надійне, ефективне і вигідне рішення для приватної хмарної інтеграції.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Охорона праці – це: система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини в процесі трудової діяльності;

Охорона праці є складовою частиною безпеки життєдіяльності [3,4].

Законом України “Про охорону праці” [3] 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» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

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.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно- обчислювальних машин» ДСанПіН 3.3.2-007-98 [5], та «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

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 негативного впливу комп'ютера на організм людини визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

Загальний нагляд за додержанням норм охорони праці покладено на прокуратуру, спеціальний покладено на професійні спілки. За безпекою контроль праці здійснюють державні й відомчі спеціалізовані інспекції.

У Законодавстві про працю міститься вимоги і норми з виробничої санітарії, техніки безпеки та норми, що регулюють робочий час, час відпочинку, звільнення та переведення на іншу роботу, а також норми праці щодо жінок, молоді, гігієнічні норми і правила, тощо.

8.2 Пожежна безпека

Пожежі в приміщеннях з оргтехнікою становлять особливу небезпеку, бо поєднані з великими матеріальними збитками. Пожежа може виникнути при взаємодії горючих речовин і джерел запалювання. Горючими речовинами є будівельні та опоряджувальні матеріали, пластмасові корпуси техніки, шнури тощо. Джерелами запалювання можуть бути електронні схеми комп'ютерів, принтерів, пристроїв електроживлення, де внаслідок різних порушень виникає

перегрівання елементів, утворюються електричні іскри та дуги, здатні спричинити займання горючих матеріалів.

З метою виявлення початкової стадії займання необхідно використовувати пристрої систем автоматичного пожежогасіння там, де цього вимагають правила пожежної безпеки.

При обслуговуванні, ремонтних та профілактичних роботах використовуються різні легкозаймисті рідини, прокладаються тимчасові електропровідники, здійснюється паяння. Виникає додаткова пожежна небезпека, яка потребує відповідних заходів пожежного захисту. До засобів гасіння пожежі, призначених для локалізації невеликих займань, належать вогнегасники, сухий пісок, азбестові ковдри. Приміщення, в який встановлено комп'ютери і де немає необхідності влаштування систем автоматичного пожежогасіння, необхідно оснащувати переносними вуглекислотними вогнегасниками з розрахунку 2 шт. на кожні 20 $м^2$ в приміщеннях. Звуковбирне облицювання стін, стель приміщень треба виконувати з негорючих та важко горючих матеріалів.

Електроустановки (можливість їх застосування, монтаж, накладка експлуатація) повинні відповідати вимогам чинних правил улаштування електроустановок, правил технічної експлуатації, електроустановок та інших нормативних документів. Ймовірність виникнення пожежі від електротехнічного та іншого одиничного виробу не повинна перевищувати 10^{-6} на рік. При короткому замиканні в місцях з’єднання проводів опір практично дорівнює нулю, звідси величина струму досягає дуже великих значень.

Персональні комп’ютери після закінчення роботи повинні відключатися від мережі не рідше 1 разу на квартал, необхідно очищати від пилу агрегати та вузли, кабельні канали та простір між підлогами. Не дозволяється розміщувати комп’ютерні зали ЕОМ у підвалах; проводити ремонт вузлів (блоків) ЕОМ безпосередньо у залах, де знаходяться персональні комп’ютери, залишати без нагляду ввімкнену в мережу електронну апаратуру, яка використовується для контролю ЕОМ.

Електричний струм силою 0,1 А є небезпечним для людини. Для попередження травм усе електричне обладнання повинне бути заземлене. Приступаючи до роботи необхідно перевірити справність обладнання, ізоляцію проводів і надійність заземлення. Доторкання до оголених струмоведучих і незахищених частин в електроустаткуванні забороняється. В разі виявлення порушень ізоляції електропроводів, відкритих струмоведучих частин електроустаткування або порушення заземлення треба негайно повідомити про це свого начальника для вжиття заходів щодо усунення несправності. Проводити самому ремонт електроустаткування забороняється.

8.3 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	2,6
Довжина	2,6
Висота	3

Таблиця 8.2 – Площа та об'єм приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6,76
Обсяг, V	$м^3$	не менше 20.0	20,3

* 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 особа. 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, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в 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.

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.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Вологість,%	Швидкість повітря, м/с	Температура,°С	Вологість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	23-24	45-55	0,11
Тепла	23-25	50-70	0,1	23-24	55-65	0,1

В літню пору застосовується кондиціонер.

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

У приміщенні знаходяться наступні джерела шуму: принтер HP 1100, 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 – це основна гігієнічна вимога. Оскільки 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.

8.4 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 значень;

Таким чином можна припустити, що основною причиною можливого 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: наочне знайомство персоналу з шляхами 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.

8.5 Розрахункова частина

Проведемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 2,6 м, довжина – 2,6 м, висота – 3 м.

У зазначеному приміщенні працює 1 особа.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F=E\cdot S\cdot K\cdot Z/n,$$

де: F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа освітлюваного приміщення (у нашому випадку S=2,6 х 2,6 = 6,76 м^2);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку K = 1,5);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку Z = 1,1);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють:

$$\rho_{стін} = 50\% \text{ і } \rho_{стелі} = 50\%.$$

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де S – площа приміщення, S = 6,76 м2;

h – розрахункова висота підвісу, h = 3 м (співпадає з висотою стелі, оскільки лампи освітлення закріплюються на стелі);

A – ширина приміщення, A = 2,6 м;

B – довжина приміщення, B = 2,6 м.

Підставимо всі значення у формулу та визначимо індекс приміщення:

i=0,43.

Знаючи індекс приміщення, знаходимо n = 0,23 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам). Підставимо всі значення у формулу, визначимо світловий потік: F=14548 Лм.

Для розрахунку будемо використовувати світлодіодні панелі LED панель 42Вт 6000К SUNLED 000000127, світловий потік яких $F_л$ = 3990 Лм.

Число ламп визначається по формулі:

$$N=F/F_л$$

де: F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначимо потрібну кількість ламп:

$$N= 14548/ 3990=3,6 \text{ шт.}$$

Приймаємо необхідну кількість ламп 4 шт.

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 умов праці, 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 випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи інтеграції для приватної хмари.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів інтеграції для приватної хмари.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем інтеграції для приватної хмари.

– Досліджена система інтеграції для приватної хмари.

– На основі отриманих результатів досліджень створена програмна реалізація системи інтеграції для приватної хмари.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання інтеграції для приватної хмари.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

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

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

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм UMAC.

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.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пятишев В.О. Дослідження та програмна реалізація системи інтеграції для приватної хмари // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2024.

2. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.

3. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.

4. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

5. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

6. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

7. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

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

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

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

11. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of

Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

12. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

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

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

15. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

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

17. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

18. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

19. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

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

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

22. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

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

24. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

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

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

27. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

28. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

29. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

30. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

31. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

32. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

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

34. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

35. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

36. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

37. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

38. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного

процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

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

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

41. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

42. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

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

44. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

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

46. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

47. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

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

49. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

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

51. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". – Випуск 5 (142). – Х.: ХУПС – 2016. – С. 148-152.

52. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". – Випуск 3 (140). – Х.: ХУПС – 2016. – С. 36-39.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи інтеграції для приватної хмари.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 19-13 від 07.08.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи інтеграції для приватної хмари.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи інтеграції для приватної хмари;

– 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 роз

робки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

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

7.2 Виконати маркетингове та економічне обґрунтування ІТ-проєкту з урахуванням цін на 3 вересня 2024 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

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 – 86 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

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 Робота над питанням охорони праці і техніки безпеки.

10.8 Маркетингове та економічне обґрунтування ІТ-проєкту.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 02.12.2024 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 18.12.2024 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

_____ Дрєєва Г.М.

Дослідження та програмна реалізація системи інтеграції для приватної хмари

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 19

Літера: РП

Кропивницький – 2024 року

Основна програма

```
import hashlib
import logging
from datetime import datetime
import random
import json
import threading
import time

# Налаштування для логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

class AuthModule:
    def __init__(self):
        self.users = {}
        self.sessions = {}

    def register_user(self, username, password):
        hashed_password = hashlib.sha256(password.encode()).hexdigest()
        self.users[username] = hashed_password
        logging.info('User registered successfully')

    def authenticate_user(self, username, password):
        hashed_password = hashlib.sha256(password.encode()).hexdigest()
        if self.users.get(username) == hashed_password:
            session_token =
hashlib.sha256(f"{username}{time.time()}".encode()).hexdigest()
            self.sessions[username] = session_token
            logging.info('User authenticated successfully')
            return session_token
        else:
            logging.warning('Authentication failed')
            return None

    def verify_session(self, username, token):
        if self.sessions.get(username) == token:
            logging.info('Session verified successfully')
            return True
        else:
            logging.warning('Session verification failed')
            return False

class DataModule:
    def __init__(self):
        self.data_storage = {}
        self.lock = threading.Lock()

    def store_data(self, key, value):
        with self.lock:
            self.data_storage[key] = value
            logging.info('Data stored successfully')

    def retrieve_data(self, key):
        with self.lock:
            data = self.data_storage.get(key, 'Data not found')
            logging.info('Data retrieved successfully')
            return data

    def delete_data(self, key):
        with self.lock:
            if key in self.data_storage:
                del self.data_storage[key]
                logging.info('Data deleted successfully')
                return True
            else:
```

```
                logging.warning('Data not found for deletion')
                return False

class ComputeModule:
    def compute_sum(self, numbers):
        result = sum(numbers)
        logging.info('Sum computed successfully')
        return result

    def compute_average(self, numbers):
        if numbers:
            result = sum(numbers) / len(numbers)
            logging.info('Average computed successfully')
            return result
        else:
            logging.warning('Empty list provided for average computation')
            return 0

    def compute_variance(self, numbers):
        mean = self.compute_average(numbers)
        variance = sum((x - mean) ** 2 for x in numbers) / len(numbers)
        logging.info('Variance computed successfully')
        return variance

class MonitoringModule:
    def __init__(self):
        self.logs = []
        self.log_count = 0

    def log_event(self, event):
        timestamp = datetime.now().isoformat()
        self.logs.append(f'{timestamp} - {event}')
        self.log_count += 1
        logging.info('Event logged successfully')

    def get_logs(self):
        logging.info('Logs retrieved successfully')
        return self.logs

    def save_logs_to_file(self, filename):
        with open(filename, 'w') as f:
            json.dump(self.logs, f)
        logging.info('Logs saved to file successfully')

class SecurityModule:
    def __init__(self):
        self.failed_attempts = {}

    def track_failed_login(self, username):
        if username not in self.failed_attempts:
            self.failed_attempts[username] = 0
        self.failed_attempts[username] += 1
        logging.warning('Failed login attempt tracked')

    def check_lockout(self, username):
        lockout = self.failed_attempts.get(username, 0) >= 3
        if lockout:
            logging.warning('User is locked out due to too many failed
attempts')
        return lockout

class CloudIntegrationSystem:
    def __init__(self):
        self.auth_module = AuthModule()
        self.data_module = DataModule()
        self.compute_module = ComputeModule()
        self.monitoring_module = MonitoringModule()
        self.security_module = SecurityModule()
```

```
    def user_registration(self, username, password):
        if self.security_module.check_lockout(username):
            return "User locked out"
        self.auth_module.register_user(username, password)

    def user_login(self, username, password):
        if self.security_module.check_lockout(username):
            return "User locked out"
        session_token = self.auth_module.authenticate_user(username, password)
        if session_token:
            return session_token
        else:
            self.security_module.track_failed_login(username)
            return "Login failed"

    def store_data_with_session(self, username, token, key, value):
        if self.auth_module.verify_session(username, token):
            self.data_module.store_data(key, value)
            self.monitoring_module.log_event(f"{username} stored data with key
{key}")
            return "Data stored"
        else:
            return "Session invalid"

    def retrieve_data_with_session(self, username, token, key):
        if self.auth_module.verify_session(username, token):
            data = self.data_module.retrieve_data(key)
            self.monitoring_module.log_event(f"{username} retrieved data with
key {key}")
            return data
        else:
            return "Session invalid"

    def delete_data_with_session(self, username, token, key):
        if self.auth_module.verify_session(username, token):
            result = self.data_module.delete_data(key)
            if result:
                self.monitoring_module.log_event(f"{username} deleted data with
key {key}")
                return "Data deleted"
            else:
                return "Data not found"
        else:
            return "Session invalid"

    def perform_computation(self, username, token, numbers):
        if self.auth_module.verify_session(username, token):
            results = {
                'sum': self.compute_module.compute_sum(numbers),
                'average': self.compute_module.compute_average(numbers),
                'variance': self.compute_module.compute_variance(numbers)
            }
            self.monitoring_module.log_event(f"{username} performed
computation")
            return results
        else:
            return "Session invalid"

    def retrieve_logs(self):
        return self.monitoring_module.get_logs()

    def save_logs(self, filename):
        self.monitoring_module.save_logs_to_file(filename)

# Запуск системи інтеграції приватної хмари
cloud_system = CloudIntegrationSystem()

# Приклад роботи з системою
cloud_system.user_registration('user1', 'password123')
```

```
session_token = cloud_system.user_login('user1', 'password123')
cloud_system.store_data_with_session('user1', session_token, 'key1', 'value1')
cloud_system.retrieve_data_with_session('user1', session_token, 'key1')
cloud_system.delete_data_with_session('user1', session_token, 'key1')
cloud_system.perform_computation('user1', session_token, [5, 10, 15, 20, 25])
cloud_system.save_logs('logs.json')

print(cloud_system.retrieve_logs())
```

Файл multi_factor_auth.py

```
import random
import logging

class MultiFactorAuth:
    def __init__(self):
        self.otp_storage = {}

    def generate_otp(self, username):
        otp = random.randint(100000, 999999)
        self.otp_storage[username] = otp
        logging.info(f"OTP generated for {username}")
        return otp  # У реальній системі OTP не повертається користувачу, а
надсилається через SMS або Email

    def verify_otp(self, username, otp):
        if self.otp_storage.get(username) == otp:
            del self.otp_storage[username]
            logging.info(f"OTP verified for {username}")
            return True
        else:
            logging.warning(f"OTP verification failed for {username}")
            return False
```

Файл data_encryption.py

```
from cryptography.fernet import Fernet
import logging

class DataEncryption:
    def __init__(self, encryption_key=None):
        self.key = encryption_key or Fernet.generate_key()
        self.cipher = Fernet(self.key)

    def encrypt_data(self, data):
        encrypted_data = self.cipher.encrypt(data.encode())
        logging.info("Data encrypted successfully")
        return encrypted_data

    def decrypt_data(self, encrypted_data):
        decrypted_data = self.cipher.decrypt(encrypted_data).decode()
        logging.info("Data decrypted successfully")
        return decrypted_data
```

Файл user_roles.py

```
import logging

class UserRoles:
    def __init__(self):
        self.user_roles = {}

    def assign_role(self, username, role):
        self.user_roles[username] = role
        logging.info(f"Role '{role}' assigned to {username}")

    def get_role(self, username):
        role = self.user_roles.get(username, 'No role assigned')
        logging.info(f"Role retrieved for {username}")
        return role

    def check_permission(self, username, required_role):
        user_role = self.user_roles.get(username)
        has_permission = user_role == required_role
        logging.info(f"Permission check for {username}: {has_permission}")
        return has_permission
```

Файл data_backup.py

```
import json
import logging
import os

class DataBackup:
    def __init__(self, backup_dir="backups"):
        self.backup_dir = backup_dir
        os.makedirs(self.backup_dir, exist_ok=True)

    def backup_data(self, data, backup_name="data_backup.json"):
        backup_path = os.path.join(self.backup_dir, backup_name)
        with open(backup_path, "w") as backup_file:
            json.dump(data, backup_file)
        logging.info("Data backup completed")

    def restore_data(self, backup_name="data_backup.json"):
        backup_path = os.path.join(self.backup_dir, backup_name)
        if os.path.exists(backup_path):
            with open(backup_path, "r") as backup_file:
                data = json.load(backup_file)
            logging.info("Data restored from backup")
            return data
        else:
            logging.warning("Backup file not found")
            return {}
```

Файл data_analytics.py

```
import statistics
import logging

class DataAnalytics:
    def calculate_mean(self, numbers):
        mean_value = statistics.mean(numbers)
        logging.info("Mean calculated successfully")
        return mean_value

    def calculate_median(self, numbers):
        median_value = statistics.median(numbers)
        logging.info("Median calculated successfully")
        return median_value

    def calculate_standard_deviation(self, numbers):
        stdev = statistics.stdev(numbers)
        logging.info("Standard deviation calculated successfully")
        return stdev
```

Файл cloud_service_integration.py

```
import logging

class CloudServiceIntegration:
    def __init__(self):
        self.integrated_services = []

    def integrate_service(self, service_name):
        if service_name not in self.integrated_services:
            self.integrated_services.append(service_name)
            logging.info(f"Service '{service_name}' integrated successfully")
        else:
            logging.warning(f"Service '{service_name}' is already integrated")

    def list_services(self):
        logging.info("Integrated services listed")
        return self.integrated_services
```

Файл auto_data_deletion.py

```
import time
import threading
import logging

class AutoDataDeletion:
    def __init__(self, data_module, ttl=3600):
        self.data_module = data_module
        self.ttl = ttl
        self.deletion_times = {}
        self.start_auto_deletion()

    def add_data_with_ttl(self, key, value):
        self.data_module.store_data(key, value)
        self.deletion_times[key] = time.time() + self.ttl
        logging.info(f"Data with key '{key}' added with TTL")

    def start_auto_deletion(self):
        def delete_expired_data():
            while True:
                current_time = time.time()
                for key, expire_time in list(self.deletion_times.items()):
                    if current_time > expire_time:
                        self.data_module.delete_data(key)
                        del self.deletion_times[key]
                time.sleep(60)  # Перевірка кожну хвилину

        thread = threading.Thread(target=delete_expired_data, daemon=True)
        thread.start()
```

Файл multi_database_support.py

```
import sqlite3
import logging

class MultiDatabaseSupport:
    def __init__(self):
        self.connections = {}

    def add_database(self, db_name, db_path):
        connection = sqlite3.connect(db_path)
        self.connections[db_name] = connection
        logging.info(f"Database '{db_name}' added with path '{db_path}'")

    def execute_query(self, db_name, query):
        if db_name in self.connections:
            cursor = self.connections[db_name].cursor()
            cursor.execute(query)
            self.connections[db_name].commit()
            logging.info(f"Query executed on database '{db_name}'")
            return cursor.fetchall()
        else:
            logging.warning(f"Database '{db_name}' not found")
            return None
```

Файл api_integration.py

```
import requests
import logging

class APIIntegration:
    def __init__(self, api_base_url):
        self.api_base_url = api_base_url
    def make_get_request(self, endpoint):
        url = f"{self.api_base_url}/{endpoint}"
        response = requests.get(url)
        if response.status_code == 200:
            logging.info(f"GET request successful for endpoint '{endpoint}'")
            return response.json()
        else:
            logging.warning(f"GET request failed for endpoint '{endpoint}'")
            return None

    def make_post_request(self, endpoint, data):
        url = f"{self.api_base_url}/{endpoint}"
        response = requests.post(url, json=data)
        if response.status_code == 200:
            logging.info(f"POST request successful for endpoint '{endpoint}'")
            return response.json()
        else:
            logging.warning(f"POST request failed for endpoint '{endpoint}'")
            return None
```

Файл parallel_request_processing.py

```
import threading
import logging

class ParallelRequestProcessor:
    def __init__(self):
        self.lock = threading.Lock()

    def process_request(self, request_function, *args):
        thread = threading.Thread(target=self._process_and_log,
args=(request_function, *args))
        thread.start()

    def _process_and_log(self, request_function, *args):
        result = request_function(*args)
        with self.lock:
            logging.info(f"Request processed with result: {result}")
        return result
```

Файл automated_reporting.py

```
import logging
import json
from datetime import datetime

class AutomatedReporting:
    def __init__(self, report_path="reports"):
        self.report_path = report_path

    def generate_report(self, data, report_name=None):
        if not report_name:
            report_name = f"report_{datetime.now().isoformat()}.json"
        report_file = f"{self.report_path}/{report_name}"
        with open(report_file, 'w') as f:
            json.dump(data, f)
        logging.info(f"Report generated: {report_file}")
        return report_file
```

Файл machine_learning_module.py

```
from sklearn.linear_model import LinearRegression
import numpy as np
import logging

class MachineLearningModule:
    def __init__(self):
        self.model = LinearRegression()

    def train_model(self, x_train, y_train):
        x_train = np.array(x_train).reshape(-1, 1)
```

```
        y_train = np.array(y_train)
        self.model.fit(x_train, y_train)
        logging.info("Model trained successfully")

    def predict(self, x_input):
        x_input = np.array(x_input).reshape(-1, 1)
        predictions = self.model.predict(x_input).tolist()
        logging.info("Prediction made successfully")
        return predictions
```

Файл performance_tracking.py

```
import time
import logging

class PerformanceTracking:
    def __init__(self):
        self.performance_data = {}

    def track_performance(self, operation_name, operation_function, *args):
        start_time = time.time()
        result = operation_function(*args)
        end_time = time.time()
        duration = end_time - start_time
        self.performance_data[operation_name] = duration
        logging.info(f"Performance tracked for {operation_name}: {duration}
seconds")
        return result

    def get_performance_data(self):
        logging.info("Performance data retrieved")
        return self.performance_data
```

Файл admin_web_interface.py

```
from flask import Flask, request, jsonify
import logging

app = Flask(__name__)

@app.route('/admin/register', methods=['POST'])
def register_user():
    data = request.json
    username = data.get("username")
    password = data.get("password")
    # Registration logic goes here
    logging.info(f"User {username} registered through admin interface")
    return jsonify({"status": "User registered"}), 200

@app.route('/admin/monitoring', methods=['GET'])
def monitoring():
    # Example response with mock data
    data = {"system_status": "Operational", "active_sessions": 42}
    logging.info("Monitoring data retrieved")
    return jsonify(data), 200

# app.run() запускати окремо для адміністративного інтерфейсу
```

Файл big_data_processing.py

```
import logging

class BigDataProcessor:
    def __init__(self):
        self.data = []

    def add_data_chunk(self, data_chunk):
        self.data.extend(data_chunk)
        logging.info("Data chunk added successfully")

    def process_data(self):
        # Mock processing function
        processed_data = {"total_items": len(self.data), "processed": True}
        logging.info("Big data processed successfully")
        return processed_data

    def clear_data(self):
        self.data = []
        logging.info("Big data storage cleared")
```

Файл log_filtering.py

```
import logging

class LogFiltering:
    def __init__(self, logs):
        self.logs = logs

    def filter_logs(self, level="INFO"):
        filtered_logs = [log for log in self.logs if log.get("level") == level]
        logging.info(f"Logs filtered by level: {level}")
        return filtered_logs

    def search_logs(self, keyword):
        filtered_logs = [log for log in self.logs if keyword in
log.get("message")]
        logging.info(f"Logs filtered by keyword: {keyword}")
        return filtered_logs
```

Файл security_compliance_check.py

```
import logging

class SecurityComplianceCheck:
    def __init__(self):
        self.compliance_rules = []

    def add_rule(self, rule):
        self.compliance_rules.append(rule)
        logging.info(f"Compliance rule added: {rule}")

    def check_compliance(self, data):
        non_compliant = []
        for rule in self.compliance_rules:
            if not rule(data):
                non_compliant.append(rule.__name__)
        if non_compliant:
            logging.warning(f"Data is non-compliant with rules:
{non_compliant}")
            return False, non_compliant
        logging.info("Data is compliant with all rules")
        return True, []
```

Файл automatic_resource_allocation.py

```
import logging

class ResourceAllocation:
    def __init__(self, total_resources):
        self.total_resources = total_resources
        self.allocated_resources = 0

    def allocate_resources(self, amount):
        if self.allocated_resources + amount <= self.total_resources:
            self.allocated_resources += amount
            logging.info(f"{amount} resources allocated successfully")
            return True
        else:
            logging.warning("Resource allocation failed - not enough resources")
            return False

    def deallocate_resources(self, amount):
        if self.allocated_resources - amount >= 0:
            self.allocated_resources -= amount
            logging.info(f"{amount} resources deallocated successfully")
            return True
        else:
            logging.warning("Resource deallocation failed - insufficient
allocated resources")
            return False

    def get_status(self):
        status = {
            "total_resources": self.total_resources,
            "allocated_resources": self.allocated_resources,
            "available_resources": self.total_resources -
self.allocated_resources
        }
        logging.info("Resource allocation status retrieved")
        return status
```