

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНКИ СЕРВІСІВ
IP-ТЕЛЕФОНІЇ
SOFTWARE TO ASSESS THE QUALITY OF
IP TELEPHONY SERVICES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-121

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ІСМАІЛОВ І.Ю.

(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ІСМАІЛОВА Іллі Юрійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для оцінки сервісів IP-телефонії. Software to Assess the Quality of IP Telephony Services
 керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
 затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209
2. Строк подання студентом проєкту (роботи) 02 червня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Модель та методи програмного забезпечення для оцінки сервісів IP-телефонії. 3. Опис програмного забезпечення. 4. Експлуатація та тестування програмного забезпечення.
5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Ілля ІСМАІЛОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 106 с., 19 табл., 37 рис., 3 дод., 24 джерела.

IP-ТЕЛЕФОНІЯ, VOIP, ЯКІСТЬ ОБСЛУГОВУВАННЯ, МОНІТОРИНГ, ОЦІНКА ЯКОСТІ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, REACT, NODE.JS, MACHINE LEARNING.

Об'єкт — процес розроблення програмного забезпечення для моніторингу та аналізу ефективності сервісів IP-телефонії.

Предмет — методи та програмні засоби автоматизованої оцінки якості сервісів IP-телефонії на основі множини метрик.

Мета роботи — підвищення точності та оперативності оцінювання сервісів IP-телефонії шляхом розроблення спеціалізованого програмного забезпечення з комплексною системою метрик та алгоритмів аналізу.

Матеріали, методи та технічні засоби: мікросервісна архітектура, мови програмування JavaScript, TypeScript та Python, фреймворки React і Node.js, контейнеризація Docker та Kubernetes, часова база даних InfluxDB, MongoDB, бібліотеки машинного навчання TensorFlow і scikit-learn.

Результати. Створено програмне забезпечення для оцінки сервісів IP-телефонії, яке забезпечує збір та аналіз мережевих метрик (затримка, джитер, втрата пакетів), обчислення показників якості голосу (MOS, R-фактор), виявлення проблем та аномалій.

Висновки. Розроблено програмне забезпечення для оцінки сервісів IP-телефонії, яке відповідає всім визначеним функціональним вимогам, забезпечує комплексний підхід до оцінювання якості, автоматизацію виявлення проблем та підтримку прийняття рішень.

Галузь використання — моніторинг та оптимізація якості VoIP-комунікацій в організаціях різного масштабу.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
106 pages, 19 tables, 37 figures, 3 appendixes, 24 sources.

IP TELEPHONY, VOIP, QUALITY OF SERVICE, MONITORING,
QUALITY ASSESSMENT, SOFTWARE, REACT, NODE.JS, MACHINE
LEARNING.

Object of study is the process of developing software for monitoring and analyzing the effectiveness of IP telephony services.

Subject of study are methods and software tools for automated quality assessment of IP telephony services based on multiple performance and reliability metrics.

The purpose of the work is to increase the accuracy and efficiency of evaluating IP telephony services by developing specialized software with a comprehensive system of metrics and analysis algorithms.

Materials, methods, and tools: microservice architecture, JavaScript, TypeScript and Python programming languages, React and Node.js frameworks, Docker and Kubernetes containerization, InfluxDB time-series database, MongoDB, TensorFlow and scikit-learn machine learning libraries, Apache Kafka messaging system.

Results. Software for assessing IP telephony services has been developed, which provides collection and analysis of network metrics (latency, jitter, packet loss), calculation of voice quality indicators (MOS, R-factor), detection of problems and anomalies, quality prediction and visualization of results through an interactive web interface.

Conclusions. Software has been developed to assess IP telephony services that meets all defined functional requirements, provides a comprehensive approach to quality assessment, automation of problem detection and decision support.

Field of use is monitoring and optimization of VoIP communications quality in organizations of various scales.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Аналіз технологій IP-телефонії	10
1.2 Методи та критерії оцінки якості сервісів IP-телефонії	14
1.3 Аналіз існуючих програмних рішень.....	20
1.4 Висновки за розділом 1	27
2 Модель та методи програмного забезпечення для оцінки сервісів IP-телефонії.....	29
2.1 Вимоги до програмного забезпечення	29
2.2 Архітектура програмного забезпечення	36
2.3 Алгоритми та методи оцінки сервісів IP-телефонії.....	43
2.4 Висновки за розділом 2	52
3 Опис програмного забезпечення	54
3.1 Структура бази даних	54
3.2 Розробка інтерфейсу програмного забезпечення	59
3.3 Визначення порядку роботи програми	61
3.4 Висновки за розділом 3	65
4 Експлуатація та тестування програмного забезпечення	67
4.1 Експлуатація програмного забезпечення	67
4.2 Тестування основних функцій програмного забезпечення	69
4.3 Висновки за розділом 4	71
Висновки	72
Перелік джерел посилання	74
Додаток А Технічне завдання	76
Додаток Б Фрагмент тексту програми	81
Додаток В Слайди презентації.....	100

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	- Application Programming Interface;
CDR	- Call Detail Records;
DSCP	- Differentiated Services Code Point;
GDPR	- General Data Protection Regulation;
HTTP	- HyperText Transfer Protocol;
HTTPS	- HyperText Transfer Protocol Secure;
IP	- Internet Protocol;
ITU-T	- International Telecommunication Union - Telecommunication;
JSON	- JavaScript Object Notation;
JWT	- JSON Web Tokens;
MOS	- Mean Opinion Score;
PESQ	- Perceptual Evaluation of Speech Quality;
POLQA	- Perceptual Objective Listening Quality Analysis;
QoS	- Quality of Service;
REST	- Representational State Transfer;
RTT	- Round-Trip Time;
SIP	- Session Initiation Protocol;
SPIT	- Spam over Internet Telephony.

ВСТУП

У сучасному світі інформаційних технологій IP-телефонія стала невід'ємною частиною комунікаційної інфраструктури підприємств та організацій різного масштабу. Технології передачі голосу через IP-мережі (Voice over IP, VoIP) забезпечують економічно ефективний та гнучкий спосіб реалізації телефонного зв'язку порівняно з традиційними телефонними системами. При цьому якість та надійність сервісів IP-телефонії безпосередньо впливають на ефективність бізнес-процесів та задоволеність користувачів [1].

З розвитком та поширенням хмарних технологій, мобільних пристроїв та зростанням вимог до якості зв'язку, оцінка сервісів IP-телефонії стає дедалі складнішим та багатогранним завданням. Існуючі програмні рішення для аналізу якості VoIP-систем часто фокусуються на окремих аспектах, таких як затримка пакетів, втрати даних або якість кодування голосу, але рідко забезпечують комплексний підхід до оцінювання [2].

Актуальність теми дослідження зумовлена необхідністю створення ефективних програмних інструментів для об'єктивної оцінки та порівняння різних сервісів IP-телефонії, що дозволить організаціям приймати обґрунтовані рішення при виборі, впровадженні та оптимізації VoIP-систем.

Об'єкт дослідження: процес розроблення програмного забезпечення для моніторингу та аналізу ефективності сервісів IP-телефонії.

Предмет дослідження: методи та програмні засоби автоматизованої оцінки якості сервісів IP-телефонії на основі множини метрик продуктивності та надійності.

Мета роботи: підвищення точності та оперативності оцінювання сервісів IP-телефонії шляхом розроблення спеціалізованого програмного забезпечення з комплексною системою метрик та алгоритмів аналізу.

Завдання дослідження:

- проаналізувати існуючі методи та програмні рішення для оцінки якості IP-телефонії, визначити їх переваги та недоліки;
- розробити систему метрик для комплексної оцінки сервісів IP-телефонії, що враховує технічні параметри та показники користувацького досвіду;
- спроектувати архітектуру програмного забезпечення для збору, обробки та аналізу даних про функціонування сервісів IP-телефонії;
- реалізувати алгоритми обчислення інтегрального показника якості на основі зібраних метрик з використанням методів машинного навчання;
- розробити інтерфейс користувача для візуалізації результатів оцінювання та підтримки прийняття рішень;
- провести тестування розробленого програмного забезпечення на реальних сервісах IP-телефонії та оцінити його ефективність.

Практична значущість роботи полягає у створенні інструментарію, який дозволить адміністраторам мереж та ІТ-менеджерам об'єктивно оцінювати якість різних сервісів IP-телефонії, оперативно виявляти проблеми та оптимізувати конфігурації для забезпечення стабільного функціонування комунікаційних систем підприємств.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз технологій IP-телефонії

IP-телефонія (Voice over Internet Protocol, VoIP) представляє собою технологію передачі голосових повідомлень через мережі з пакетною комутацією на основі протоколу IP. На відміну від традиційної телефонії, яка використовує комутацію каналів, IP-телефонія перетворює аналоговий голосовий сигнал у цифровий формат, розбиває його на пакети даних і передає через IP-мережу з подальшим перетворенням назад у голосовий сигнал на приймальній стороні [1]. Основною перевагою даної технології є суттєве зниження вартості міжнародних і міжміських дзвінків, а також можливість інтеграції з іншими IP-сервісами.

Технології IP-телефонії базуються на стеку протоколів TCP/IP, при цьому для передачі голосових даних найчастіше використовується протокол UDP (User Datagram Protocol), який забезпечує меншу затримку порівняно з TCP, хоча й не гарантує доставку всіх пакетів. Архітектура типової системи IP-телефонії включає кілька ключових компонентів, взаємодія яких показана на рисунку 1.1 [2].

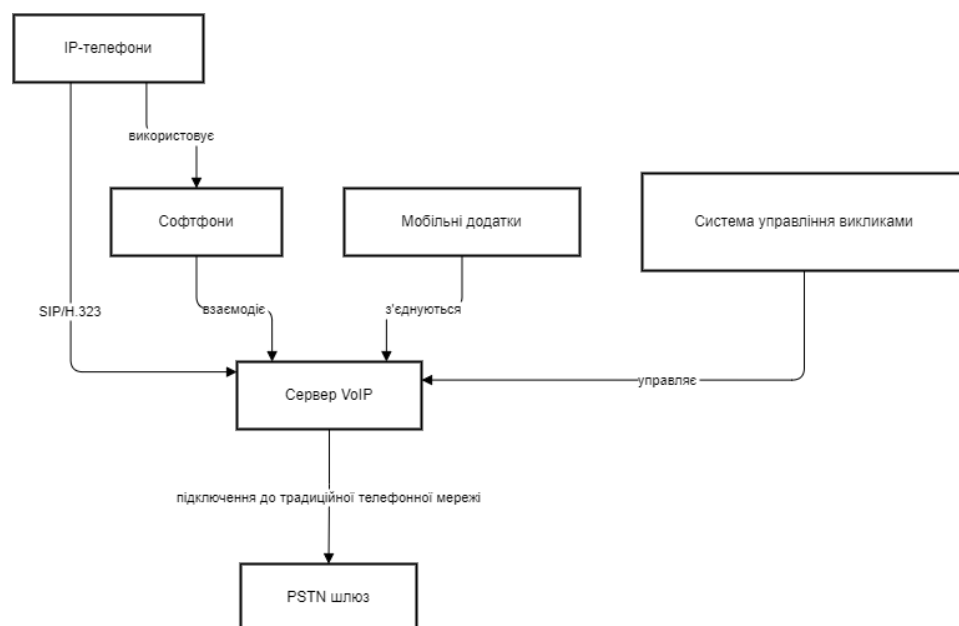


Рисунок 1.1 - Архітектура типової системи IP-телефонії [2]

В сучасних системах IP-телефонії використовуються різні протоколи сигналізації для встановлення, підтримки та завершення з'єднань. Найпоширенішими з них є SIP (Session Initiation Protocol) та H.323. SIP є більш гнучким і легким протоколом, що базується на текстових повідомленнях, подібних до HTTP, тоді як H.323 представляє собою комплексний набір протоколів, розроблений ITU-T. В таблиці 1.1 наведено порівняння основних характеристик цих протоколів [3].

Таблиця 1.1 - Порівняння протоколів сигналізації в IP-телефонії [3]

Характеристика	SIP	H.323
Розробник	IETF	ITU-T
Тип повідомлень	Текстовий	Бінарний
Складність	Середня	Висока
Масштабованість	Висока	Середня
Гнучкість	Висока	Середня
Підтримка мультимедіа	Так	Так
Поширеність	Дуже висока	Середня
Інтеграція з вебсервісами	Проста	Складна

Процес встановлення з'єднання в SIP-мережі включає кілька етапів: реєстрацію клієнта на SIP-сервері, пошук абонента, встановлення сесії, обмін медіаданими та завершення з'єднання. Діаграма послідовності типового SIP-дзвінка представлена на рисунку 1.2, яка демонструє всі етапи взаємодії між клієнтами та сервером [4].

Для кодування та декодування голосу в IP-телефонії використовуються різні кодеки, які відрізняються якістю звуку, вимогами до пропускної здатності та обчислювальними ресурсами [5]. Найпоширеніші кодеки включають: G.711 (висока якість, але висока пропускна здатність близько 64 кбіт/с); G.729 (хороша якість при низькій пропускній здатності 8

кбіт/с); Opus (адаптивний кодек із відмінною якістю при різних бітрейтах); G.722 (HD-звук з шириною смуги до 7 кГц). Вибір кодека залежить від доступної пропускної здатності мережі, обчислювальних ресурсів обладнання та вимог до якості звуку.

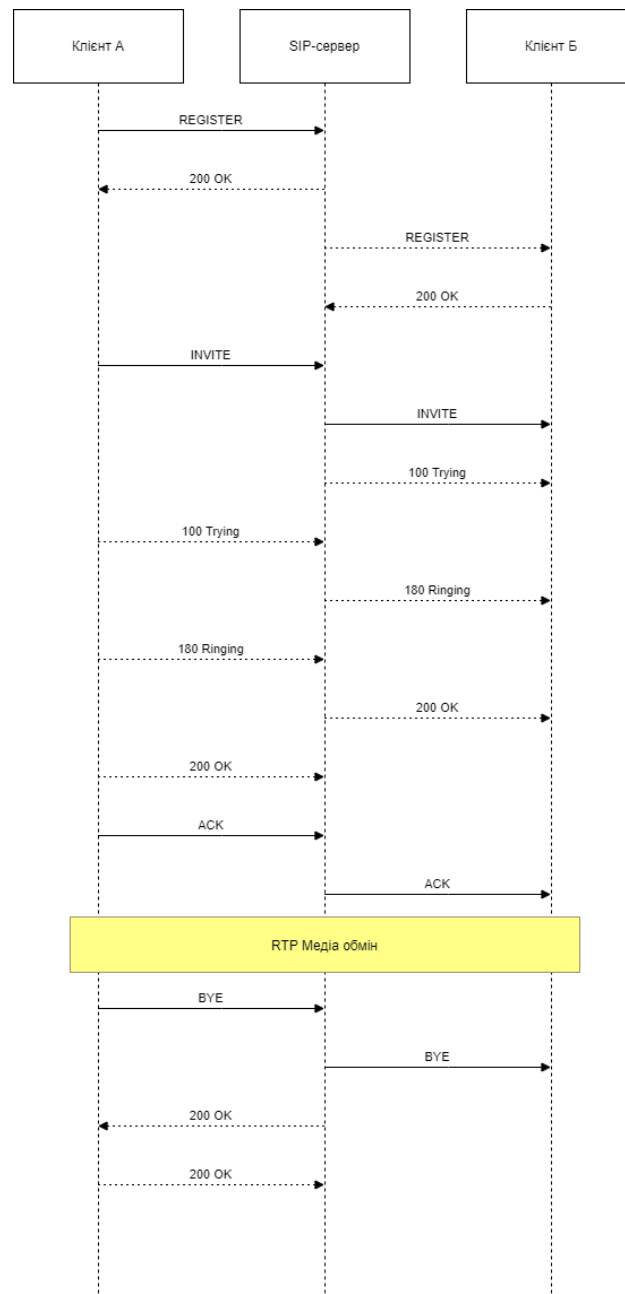


Рисунок 1.2 - Діаграма послідовності SIP-дзвінка [4]

Сучасні сервіси IP-телефонії можна класифікувати за кількома критеріями: за способом розгортання (on-premises, хмарні, гібридні); за цільовою аудиторією (корпоративні, провайдерські, споживчі); за

функціональністю (базова телефонія, уніфіковані комунікації, контакт-центри). Хмарні сервіси IP-телефонії (UCaaS - Unified Communications as a Service) стають все більш популярними завдяки гнучкості, масштабованості та низькій вартості експлуатації. Архітектура типової хмарної системи IP-телефонії показана на рисунку 1.3 [6].

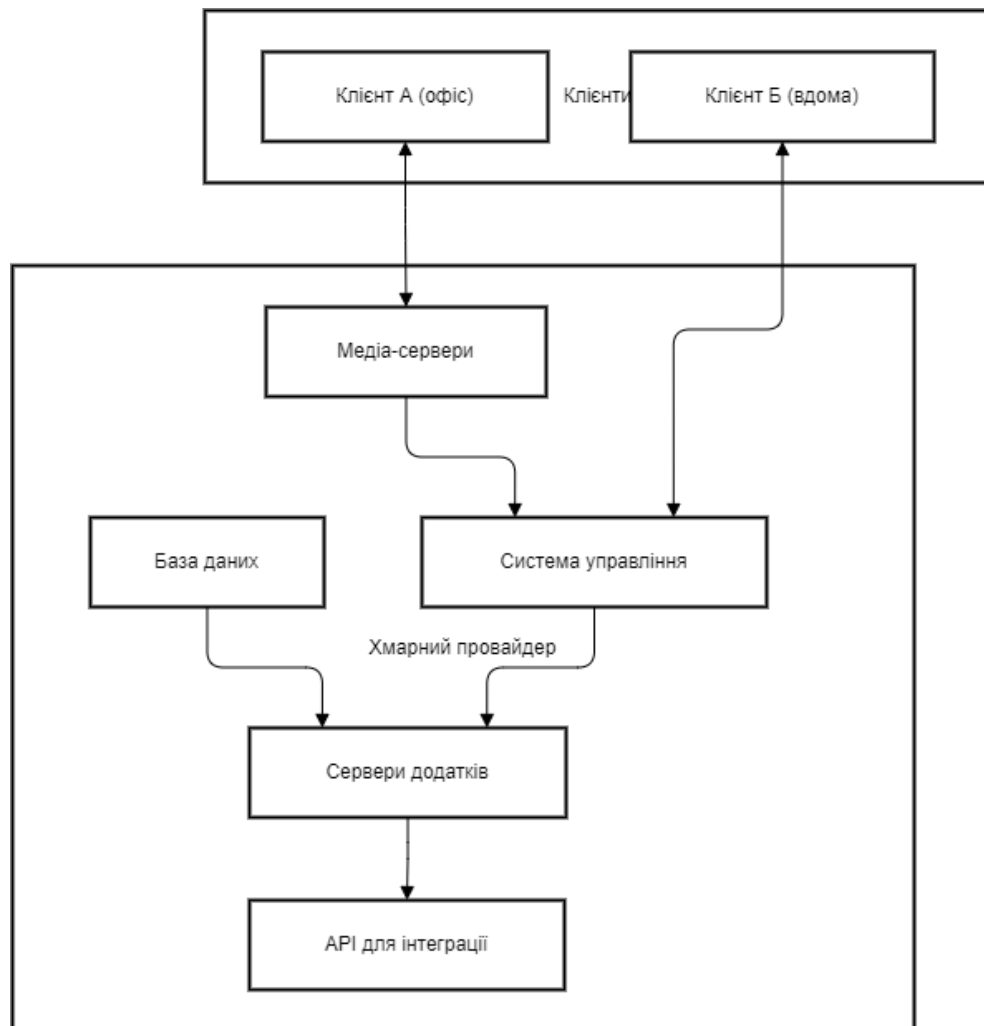


Рисунок 1.3 - Архітектура хмарної системи IP-телефонії [6]

Важливим аспектом IP-телефонії є забезпечення якості обслуговування (Quality of Service, QoS), яке включає низку механізмів для пріоритетизації голосового трафіку та забезпечення стабільної роботи в умовах обмеженої пропускної здатності [7]. Основними параметрами QoS для IP-телефонії є: затримка передачі пакетів (має не перевищувати 150 мс для комфортного спілкування); джитер (варіація затримки, оптимально

менше 30 мс); втрата пакетів (допустимо до 1% для більшості кодеків). Для забезпечення QoS використовуються технології маркування пакетів (DSCP, ToS), пріоритизації черг (QoS Queuing), резервування пропускну здатності (RSVP), а також технології формування трафіку.

У корпоративному сегменті IP-телефонія часто інтегрується з системами уніфікованих комунікацій (UC) та корпоративними додатками [8]. Уніфіковані комунікації об'єднують різні канали зв'язку (голос, відео, миттєві повідомлення, електронну пошту) в єдиному інтерфейсі, що покращує співпрацю між співробітниками. Інтеграція з CRM та ERP системами дозволяє автоматизувати бізнес-процеси та підвищити ефективність роботи.

Безпека є одним із ключових аспектів впровадження та експлуатації сервісів IP-телефонії [9]. Основні загрози включають перехоплення голосового трафіку, несанкціонований доступ до системи, DoS-атаки, SPIT (спам у IP-телефонії) та фішинг. Для захисту від цих загроз використовуються різні методи: шифрування медіатрафіку (SRTP, ZRTP); шифрування сигнального трафіку (TLS для SIP); автентифікація користувачів (DIGEST, OAuth); віртуальні приватні мережі (VPN); системи виявлення вторгнень (IDS); міжмережеві екрани, спеціалізовані для VoIP-трафіку.

1.2 Методи та критерії оцінки якості сервісів IP-телефонії

Якість сервісів IP-телефонії є багатовимірною характеристикою, що охоплює технічні параметри мережі, якість передачі голосу та суб'єктивне сприйняття користувачів. У цьому розділі розглянемо основні методи оцінки якості, стандартизовані метрики та підходи до тестування сервісів IP-телефонії.

Оцінка якості сервісів IP-телефонії може проводитися на різних рівнях: мережевому рівні, де аналізуються параметри передачі пакетів; рівні медіапотoku, де оцінюється якість кодування та декодування голосу; рівні сприйняття користувачем, де оцінюється загальне враження від сервісу [10].

Комплексний підхід до оцінки повинен враховувати всі ці аспекти, оскільки проблеми на будь-якому рівні можуть негативно вплинути на загальну якість послуги.

На мережевому рівні ключовими метриками є затримка (latency), джитер (jitter), втрата пакетів (packet loss) та пропускна здатність (bandwidth) [11]. Затримка визначається як час, необхідний для передачі пакета від джерела до пункту призначення, і вимірюється в мілісекундах. Для комфортного голосового спілкування максимально допустима затримка становить 150 мс в один бік, при цьому оптимальними вважаються значення до 100 мс. Джитер представляє собою варіацію затримки для різних пакетів і також вимірюється в мілісекундах. Високі значення джитера (понад 30 мс) можуть призвести до спотворення голосу та появи "зависань" у розмові. Втрата пакетів вимірюється у відсотках і для більшості голосових кодеків не повинна перевищувати 1%. Пропускна здатність залежить від використовуваного кодека і варіюється від 8 кбіт/с для кодека G.729 до 64 кбіт/с для G.711.

Для оцінки якості голосу в IP-телефонії розроблено декілька стандартизованих методів, найбільш поширеними з яких є MOS (Mean Opinion Score), PESQ (Perceptual Evaluation of Speech Quality), POLQA (Perceptual Objective Listening Quality Analysis) та E-модель (табл. 1.2). MOS є суб'єктивною оцінкою, що базується на експертних оцінках групи людей, які слухають зразки мовлення та оцінюють їх за шкалою від 1 (погана якість) до 5 (відмінна якість). Однак через суб'єктивність та трудомісткість MOS часто замінюється об'єктивними алгоритмічними методами. Метод PESQ (стандарт ITU-T P.862) порівнює оригінальний сигнал з сигналом, який пройшов через систему зв'язку, та обчислює показник якості. POLQA (стандарт ITU-T P.863) є вдосконаленням PESQ і краще працює з HD-звуком та сучасними кодексами. E-модель (стандарт ITU-T G.107) оцінює якість на основі мережевих параметрів і характеристик кодека, обчислюючи R-фактор, який потім може бути конвертований у приблизне значення MOS [12].

Таблиця 1.2 - Співвідношення між MOS, R-фактором та сприйняттям якості користувачем [12]

MOS	R-фактор	Якість розмови	Рівень задоволеності користувачів
4.3-5.0	90-100	Відмінна	Дуже задоволені
4.0-4.3	80-90	Добра	Задоволені
3.6-4.0	70-80	Прийнятна	Деякі користувачі незадоволені
3.1-3.6	60-70	Низька	Багато користувачів незадоволені
2.6-3.1	50-60	Погана	Майже всі користувачі незадоволені
1.0-2.6	0-50	Неприйнятна	Неприйнятно для всіх користувачів

Для комплексної оцінки якості сервісів IP-телефонії важливо аналізувати не лише якість голосу, але й інші аспекти, такі як надійність сервісу, функціональність, зручність використання та безпека [13]. Надійність характеризується коефіцієнтом доступності (availability), середнім часом між відмовами (MTBF) та часом відновлення після відмови (MTTR). Функціональність оцінюється за наявністю та якістю реалізації різних функцій, таких як переадресація, конференц-зв'язок, голосова пошта, інтеграція з іншими системами тощо. Зручність використання включає оцінку інтерфейсу користувача, простоту налаштування та управління. Безпека оцінюється за рівнем захисту від несанкціонованого доступу, шифрування даних та стійкості до різних типів атак.

Методи тестування сервісів IP-телефонії можна розділити на активні та пасивні. Активні методи передбачають генерацію тестового трафіку та вимірювання параметрів його передачі. Пасивні методи базуються на аналізі реального трафіку без втручання в роботу системи. Обидва підходи мають свої переваги та недоліки, які представлені на рисунку 1.4 [14].

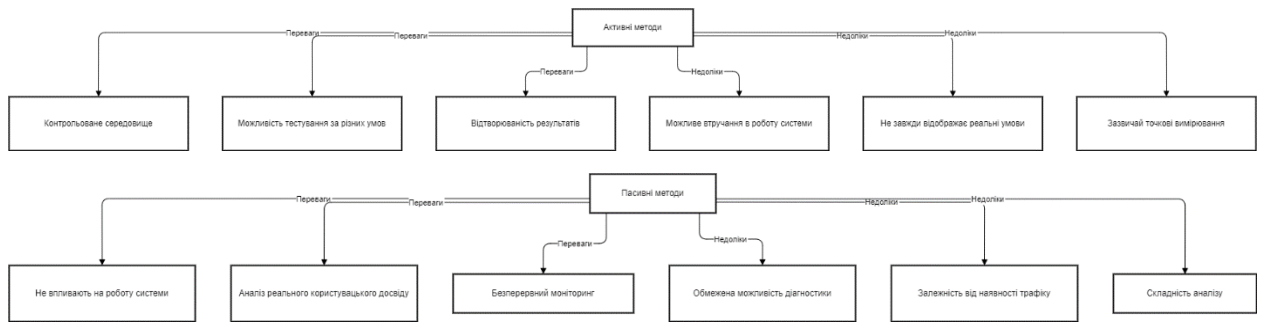


Рисунок 1.4 - Порівняння активних і пасивних методів тестування IP-телефонії [14]

Для активного тестування використовуються різні інструменти, такі як генератори голосового трафіку, симулятори мережевих умов та аналізатори протоколів. Генератори трафіку дозволяють створювати різні типи навантаження на систему, імітуючи різну кількість одночасних дзвінків з різними характеристиками. Симулятори мережевих умов дозволяють тестувати роботу системи в умовах обмеженої пропускну здатності, підвищеної затримки, джитера та втрати пакетів. Аналізатори протоколів дозволяють детально вивчити процес встановлення з'єднання, передачі даних та завершення викликів [15].

Для пасивного моніторингу використовуються системи аналізу мережевого трафіку та збору статистики з компонентів системи IP-телефонії. Аналіз трафіку здійснюється за допомогою технології зеркалювання портів (port mirroring) або через мережеві TAP-пристрої, які дозволяють перехоплювати трафік без втручання в його передачу. Зібраний трафік аналізується спеціалізованим програмним забезпеченням, яке виділяє VoIP-пакети, декодує протоколи сигналізації та обчислює різні метрики якості. Крім того, багато систем IP-телефонії надають API для збору статистики з серверів, шлюзів та IP-телефонів, що дозволяє отримувати дані про якість сервісу безпосередньо від компонентів системи [16].

Для повноцінної оцінки якості сервісів IP-телефонії в реальних умовах експлуатації можуть застосовуватися методи синтетичного моніторингу та аналізу агрегованих даних про дзвінки (CDR - Call Detail

Records). Синтетичний моніторинг передбачає періодичне виконання автоматизованих тестових дзвінків між різними точками мережі та вимірювання їх параметрів. Це дозволяє виявляти проблеми до того, як вони вплинуть на користувачів. Аналіз CDR дозволяє виявляти закономірності та тренди в якості сервісу, ідентифікувати проблемні ділянки мережі та оптимізувати конфігурацію системи [17].

Комплексний підхід до оцінки якості сервісів IP-телефонії повинен включати набір метрик, які охоплюють різні аспекти роботи системи. На рисунку 1.5 представлена ієрархія метрик, які можуть бути використані для оцінки сервісів IP-телефонії [18].

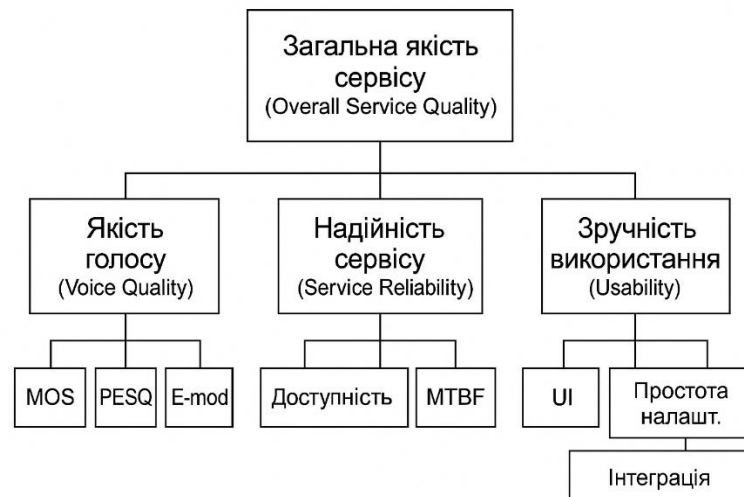


Рисунок 1.5 - Ієрархія метрик оцінки сервісів IP-телефонії [18]

При розробці програмного забезпечення для оцінки сервісів IP-телефонії важливо враховувати різні сценарії використання та типи користувачів [19]. Адміністратори мереж потребують детальної технічної інформації про роботу системи, аналітики - агрегованих даних для виявлення трендів та прогнозування проблем, а керівники - узагальнених показників для прийняття бізнес-рішень. Для задоволення потреб різних типів користувачів програмне забезпечення повинно забезпечувати різні рівні деталізації даних та відповідний користувацький інтерфейс.

У таблиці 1.3 представлено типові проблеми IP-телефонії, їх вплив на якість сервісу та методи виявлення [19].

Таблиця 1.3 - Типові проблеми IP-телефонії та методи їх виявлення [19]

Проблема	Вплив на якість	Методи виявлення
Висока затримка	Затримка відповіді, ефект "луни"	Вимірювання RTT, тести затримки
Джитер	Переривання, спотворення голосу	Аналіз варіації затримки пакетів
Втрата пакетів	Пропуски в розмові, спотворення	Порівняння відправлених/отриманих пакетів
Недостатня пропускна здатність	Низька якість, обриви зв'язку	Моніторинг використання смуги пропускання
Проблеми з кодеком	Спотворення голосу	Аналіз MOS, PESQ
Проблеми сигналізації	Неможливість встановлення з'єднання	Аналіз логів сервера, захоплення SIP-трафіку
Проблеми з безпекою	Несанкціонований доступ, відмова в обслуговуванні	Моніторинг безпеки, аналіз аномалій

Сучасні підходи до оцінки якості сервісів IP-телефонії все частіше включають методи машинного навчання та штучного інтелекту для автоматичного виявлення проблем, прогнозування відмов та оптимізації конфігурації. Алгоритми класифікації можуть використовуватися для категоризації проблем на основі набору симптомів, алгоритми кластеризації - для виявлення груп пов'язаних проблем, а алгоритми регресії - для прогнозування майбутніх значень метрик якості. На рисунку 1.6 представлена архітектура системи оцінки якості IP-телефонії з використанням методів машинного навчання [20].

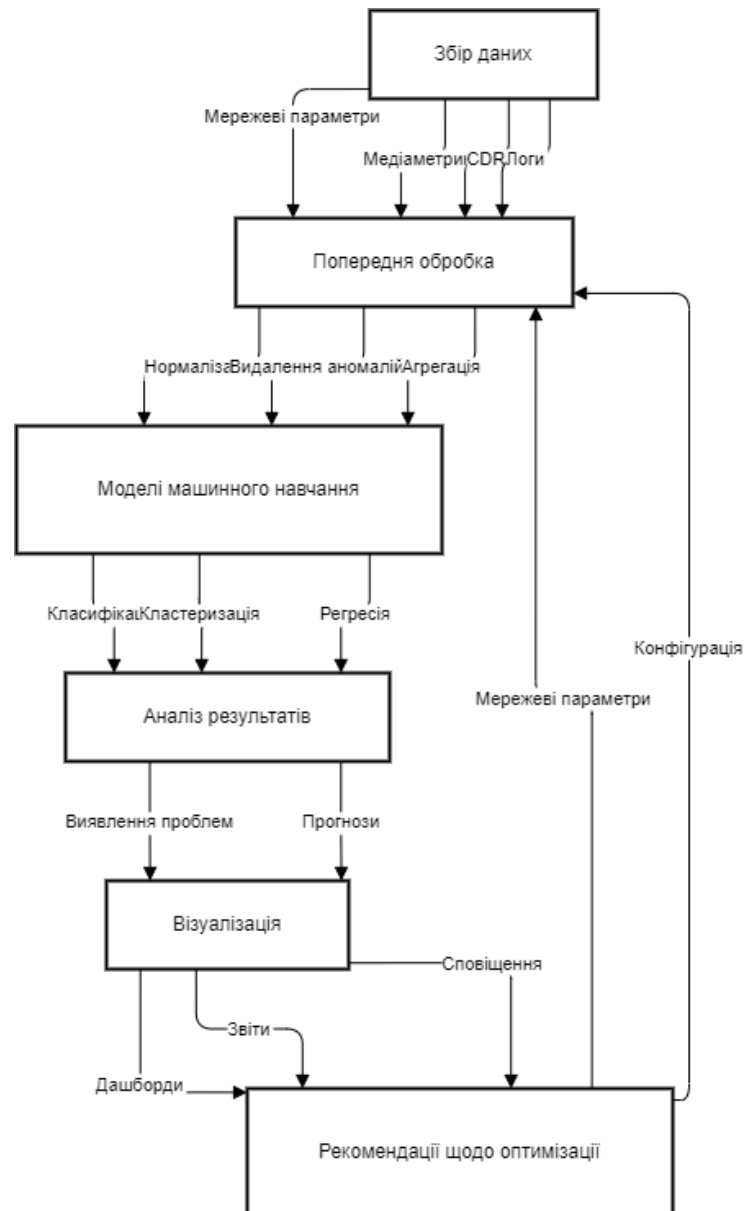


Рисунок 1.6 - Архітектура системи оцінки якості з використанням машинного навчання [20]

1.3 Аналіз існуючих програмних рішень

Ринок програмного забезпечення для оцінки та моніторингу сервісів IP-телефонії представлений широким спектром рішень – від спеціалізованих інструментів для тестування окремих параметрів до комплексних систем моніторингу. У цьому розділі проведемо аналіз найбільш поширених інструментів, порівняємо їх функціональні можливості та виявимо

обмеження, які необхідно подолати при розробці нового програмного забезпечення.

Існуючі програмні рішення для оцінки сервісів IP-телефонії можна класифікувати за кількома критеріями: за призначенням (для тестування, моніторингу, діагностики); за типом оцінюваних параметрів (мережеві параметри, якість голосу, продуктивність сервера); за способом розгортання (автономні додатки, хмарні сервіси, вбудовані системи). На рисунку 1.7 представлена класифікація програмних рішень для оцінки сервісів IP-телефонії [21].

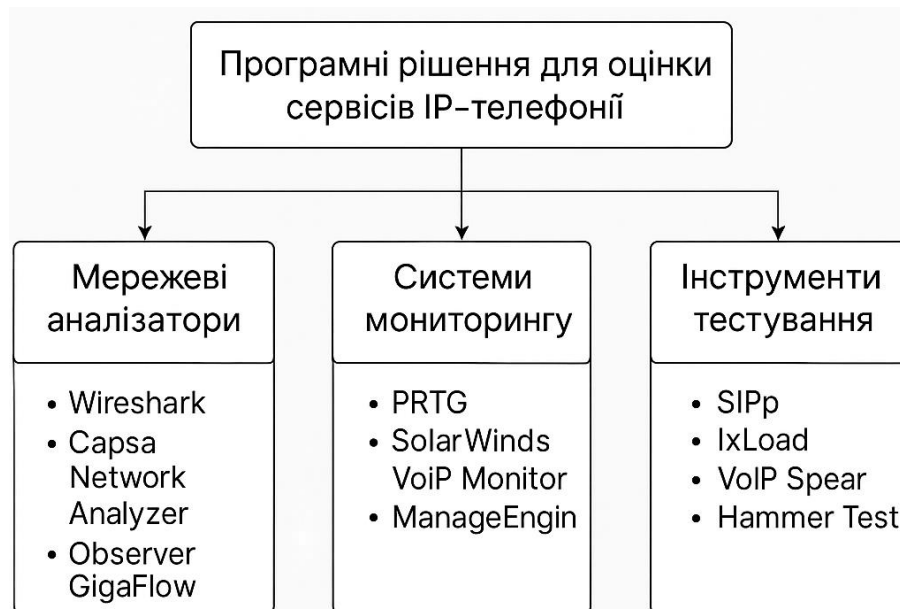


Рисунок 1.7 - Класифікація програмних рішень для оцінки сервісів IP-телефонії [21]

Одним з найбільш поширених інструментів для аналізу мережевого трафіку IP-телефонії є Wireshark – безкоштовний аналізатор протоколів з відкритим вихідним кодом. Wireshark дозволяє захоплювати та аналізувати пакети в реальному часі, підтримує декодування основних протоколів VoIP (SIP, RTP, H.323) та надає засоби для аналізу якості голосового потоку. Основними перевагами Wireshark є гнучкість, наявність потужних фільтрів, висока деталізація даних та можливість розширення функціональності за

допомогою плагінів. Однак, Wireshark має і обмеження: складність використання для нетехнічних користувачів; відсутність автоматизованого аналізу великих обсягів даних; необхідність ручного налаштування для отримання кількісних показників якості [22].

Спеціалізовані системи моніторингу VoIP, такі як SolarWinds VoIP & Network Quality Manager, PRTG Network Monitor, ManageEngine VoIP Monitor, забезпечують безперервний моніторинг роботи систем IP-телефонії та автоматичне виявлення проблем. Ці системи збирають дані з різних джерел (мережеве обладнання, IP-телефони, сервери VoIP), агрегують їх та представляють у вигляді інформативних дашбордів та звітів. У таблиці 1.4 наведено порівняння функціональних можливостей найбільш популярних систем моніторингу VoIP [23].

Таблиця 1.4 - Порівняння систем моніторингу VoIP [23]

Функціональність	SolarWinds VoIP & Network Quality Manager	PRTG Network Monitor	ManageEngine VoIP Monitor	Observer Platform
Моніторинг якості дзвінків	Так	Так	Так	Так
Аналіз сигнального трафіку	Так	Обмежений	Так	Так
Аналіз медіатрафіку	Так	Так	Так	Так
Синтетичні тести	Так	Ні	Обмежені	Так
Аналіз CDR	Так	Ні	Так	Обмежений
Карта мережі	Так	Так	Так	Так
Оповіщення	Так	Так	Так	Так
Звіти	Розширені	Базові	Розширені	Розширені
Інтеграція з ITSM	Так	Обмежена	Так	Обмежена
Прогностична аналітика	Обмежена	Ні	Обмежена	Так

Для тестування продуктивності та функціональності систем IP-телефонії використовуються спеціалізовані інструменти, такі як SIPp, IxLoad, VoIP Spear, Hammer Test [24]. SIPp є інструментом з відкритим вихідним кодом для генерації SIP-трафіку та тестування продуктивності SIP-серверів. IxLoad та Hammer Test є комерційними рішеннями для комплексного тестування систем IP-телефонії, які дозволяють емулювати тисячі одночасних дзвінків та різні сценарії використання. VoIP Spear фокусується на оцінці якості голосу та надає засоби для вимірювання MOS в різних мережеских умовах.

Для оцінки якості голосу в IP-телефонії використовуються такі інструменти, як VQmon, POLQA, ITU-T P.563 [5]. VQmon реалізує алгоритми обчислення R-фактора (модель E) та MOS на основі мережеских параметрів без аналізу самого голосового потоку. POLQA є стандартизованим алгоритмом для порівняння оригінального та перетвореного голосових сигналів та обчислення об'єктивної оцінки якості. ITU-T P.563 дозволяє оцінювати якість голосу без еталонного сигналу, аналізуючи лише отриманий сигнал.

Важливим класом програмних рішень є інструменти для діагностики та вирішення проблем з якістю IP-телефонії, такі як VoIP Troubleshooter, PathSolutions TotalView, Metricell VoIP Assurance [6]. Ці інструменти допомагають ідентифікувати причини проблем з якістю, локалізувати їх у мережі та надати рекомендації щодо їх усунення. Вони часто включають функції автоматичного тестування мережі, аналізу конфігурації обладнання та візуалізації проблемних ділянок.

Для оцінки роботи хмарних сервісів IP-телефонії існують спеціалізовані рішення, такі як ThousandEyes, AppNeta, Exoprise CloudReady [7]. Ці інструменти дозволяють моніторити доступність та продуктивність хмарних сервісів з різних географічних локацій, оцінювати якість з'єднання між користувачами та хмарними провайдерами, та виявляти проблеми на різних ділянках мережі.

Архітектура типової системи моніторингу IP-телефонії включає кілька основних компонентів, які взаємодіють між собою для збору, обробки та візуалізації даних про якість сервісу. На рисунку 1.8 представлена типова архітектура такої системи [8].

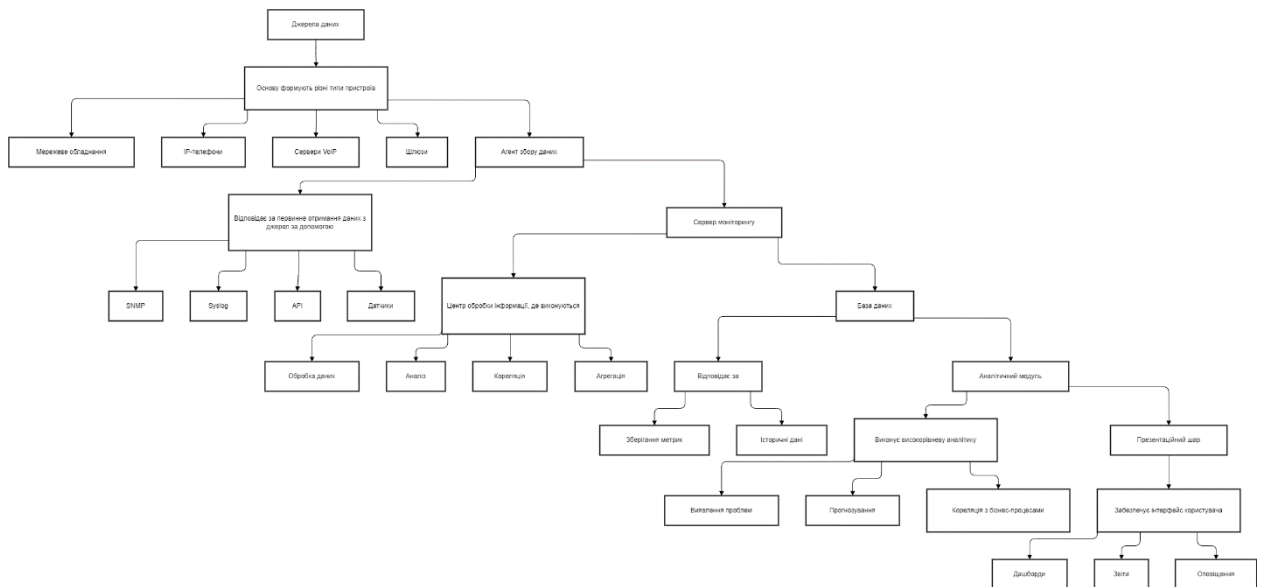


Рисунок 1.8 - Архітектура типової системи моніторингу IP-телефонії [8]

Незважаючи на різноманітність доступних рішень, більшість існуючих програмних продуктів для оцінки сервісів IP-телефонії мають певні обмеження [9]: фокус на окремих аспектах оцінки без комплексного підходу; складність інтеграції різних інструментів для отримання повної картини; недостатнє використання методів машинного навчання для прогнозування проблем та оптимізації параметрів; обмежені можливості для порівняльного аналізу різних сервісів IP-телефонії; висока вартість комерційних рішень при обмеженій функціональності безкоштовних альтернатив.

Аналіз функціональних можливостей існуючого програмного забезпечення дозволяє визначити ключові вимоги до нового рішення для оцінки сервісів IP-телефонії: комплексний підхід, що охоплює різні аспекти оцінки; автоматизація процесів збору та аналізу даних; гнучкість у налаштуванні метрик та параметрів оцінки; наявність інтуїтивного інтерфейсу для користувачів різного рівня технічної підготовки; можливість

інтеграції з існуючими системами моніторингу та управління; використання методів машинного навчання для прогнозування проблем та оптимізації параметрів; забезпечення належного рівня безпеки та конфіденційності даних.

У таблиці 1.5 представлено порівняння вимог різних груп користувачів до програмного забезпечення для оцінки сервісів IP-телефонії [10].

Таблиця 1.5 - Вимоги різних груп користувачів до ПЗ для оцінки сервісів IP-телефонії

Група користувачів	Основні задачі	Ключові вимоги
Адміністратори мереж	Моніторинг, діагностика, оптимізація	Детальні технічні метрики, засоби діагностики, інтеграція з системами управління
ІТ-менеджери	Планування, порівняльний аналіз, бюджетування	Агреговані метрики, порівняльний аналіз, звіти, прогнози
Керівники	Прийняття стратегічних рішень	Бізнес-метрики, аналіз тенденцій, інтеграція з бізнес-процесами
Оператори контакт-центрів	Забезпечення якості обслуговування	Моніторинг у реальному часі, сповіщення, інтеграція з CRM
Розробники VoIP-додатків	Тестування, оптимізація, інтеграція	API, автоматизовані тести, інструменти для навантажувального тестування

Важливим аспектом розробки нового програмного забезпечення є вибір технологій та архітектурних рішень [1]. Сучасні тенденції в розробці систем моніторингу та оцінки якості ІТ-сервісів включають: використання мікросервісної архітектури для забезпечення гнучкості та масштабованості; застосування контейнеризації для спрощення розгортання та управління; використання часових баз даних (time-series databases) для ефективного

зберігання та аналізу метрик; впровадження методів машинного навчання для прогностичної аналітики; використання сучасних технологій візуалізації для створення інтерактивних дашбордів.

На рисунку 1.9 представлена пропонована архітектура програмного забезпечення для оцінки сервісів IP-телефонії з урахуванням сучасних технологічних тенденцій та виявлених недоліків існуючих рішень.

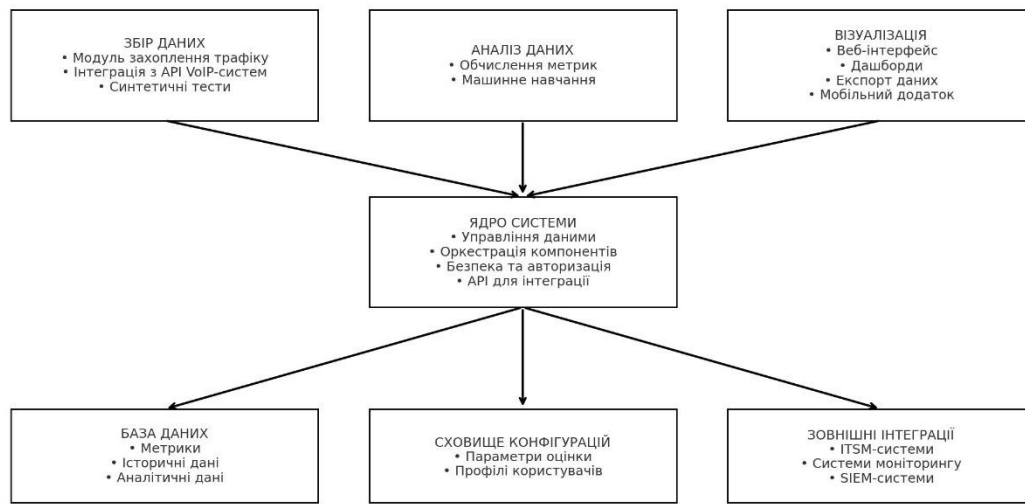


Рисунок 1.9 - Пропонована архітектура ПЗ для оцінки сервісів IP-телефонії

Підсумовуючи аналіз існуючих програмних рішень та вимог до нової системи, можна виділити такі ключові функціональні блоки, які повинні бути реалізовані в розроблюваному програмному забезпеченні: модуль аналізу мережевого трафіку для оцінки мережевих параметрів; модуль оцінки якості голосу з використанням різних методів (PESQ, POLQA, E-модель); модуль тестування продуктивності системи IP-телефонії; модуль аналізу CDR для виявлення тенденцій та аномалій; модуль машинного навчання для прогнозування проблем та оптимізації параметрів; модуль порівняльного аналізу різних сервісів IP-телефонії; модуль візуалізації результатів у вигляді інтерактивних дашбордів та звітів.

1.4 Висновки за розділом 1

У першому розділі було проведено комплексний аналіз технологій IP-телефонії, методів оцінки їх якості та існуючих програмних рішень, що дозволило сформулювати теоретичну основу для розробки нового програмного забезпечення.

В результаті дослідження було визначено, що основними технологіями IP-телефонії є протоколи сигналізації SIP та H.323, медіа-протоколи RTP/RTCP, а також різноманітні кодеки для стиснення голосу. Встановлено, що сучасні тенденції розвитку IP-телефонії включають інтеграцію з системами уніфікованих комунікацій, перехід до хмарних рішень та забезпечення мобільного доступу, що підвищує вимоги до якості та надійності цих сервісів.

Аналіз методів та критеріїв оцінки якості сервісів IP-телефонії показав, що для комплексної оцінки необхідно враховувати мережеві параметри (затримка, джитер, втрата пакетів), якість голосу (MOS, PESQ, POLQA), надійність сервісу та зручність використання. Виявлено, що ефективна оцінка вимагає поєднання активних і пасивних методів тестування, а також застосування методів машинного навчання для автоматичного виявлення проблем та прогнозування якості.

Дослідження існуючих програмних рішень для оцінки сервісів IP-телефонії дозволило виявити їх обмеження: фокус на окремих аспектах без комплексного підходу; складність інтеграції різних інструментів; недостатнє використання методів машинного навчання; обмежені можливості для порівняльного аналізу різних сервісів. На основі цього аналізу сформульовано вимоги до нової системи, серед яких: комплексний підхід до оцінки; автоматизація процесів збору та аналізу даних; гнучкість у налаштуванні метрик; інтуїтивний інтерфейс для різних груп користувачів; можливість інтеграції з існуючими системами; використання методів машинного навчання.

Запропоновано архітектуру програмного забезпечення на основі мікросервісного підходу, яка включає модулі збору даних, аналізу даних та візуалізації. Така архітектура забезпечує гнучкість, масштабованість та можливість інтеграції з існуючими системами моніторингу та управління.

Таким чином, проведений аналіз створив теоретичну основу для розробки програмного забезпечення, яке дозволить підвищити точність та оперативність оцінювання сервісів IP-телефонії шляхом комплексного аналізу різних аспектів їх роботи з використанням сучасних технологій та підходів.

2 МОДЕЛЬ ТА МЕТОДИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНКИ СЕРВІСІВ ІР-ТЕЛЕФОНІЇ

2.1 Вимоги до програмного забезпечення

На основі проведеного аналізу технологій ІР-телефонії, методів оцінки їх якості та існуючих програмних рішень, у цьому розділі сформулюємо детальні вимоги до розроблюваного програмного забезпечення. Ці вимоги будуть слугувати основою для проєктування архітектури системи та розробки її компонентів.

Функціональні вимоги до програмного забезпечення для оцінки сервісів ІР-телефонії можна розділити на кілька основних категорій: збір даних, аналіз та обробка даних, візуалізація результатів, адміністрування та інтеграція з іншими системами [2]. Розглянемо кожную категорію детальніше.

Функціональні вимоги до підсистеми збору даних включають можливості захоплення та аналізу мережевого трафіку, взаємодії з API різних систем ІР-телефонії, проведення активних тестів та збору статистики з мережевого обладнання. Система повинна підтримувати різні протоколи сигналізації (SIP, H.323) та медіа-протоколи (RTP, RTCP), а також забезпечувати можливість збору даних з географічно розподілених точок моніторингу. У таблиці 2.1 представлено детальний перелік функціональних вимог до підсистеми збору даних.

Функціональні вимоги до підсистеми аналізу та обробки даних включають обчислення різних метрик якості, виявлення проблем та аномалій, прогнозування якості та генерацію рекомендацій щодо оптимізації. Ця підсистема повинна забезпечувати комплексний аналіз зібраних даних з використанням різних алгоритмів та методів, включаючи методи машинного навчання.

Таблиця 2.1 - Функціональні вимоги до підсистеми збору даних

Код вимоги	Опис вимоги	Пріоритет
FR-DC-01	Захоплення та декодування SIP-трафіку	Високий
FR-DC-02	Захоплення та декодування RTP-трафіку	Високий
FR-DC-03	Підтримка протоколу H.323	Середній
FR-DC-04	Інтеграція з API VoIP-систем для отримання CDR	Високий
FR-DC-05	Збір статистики з мережевого обладнання через SNMP	Середній
FR-DC-06	Проведення синтетичних тестів якості дзвінків	Високий
FR-DC-07	Збір даних з географічно розподілених точок	Середній
FR-DC-08	Підтримка різних кодеків (G.711, G.729, Opus, та ін.)	Високий
FR-DC-09	Збір даних про QoS-параметри мережі	Високий
FR-DC-10	Інтеграція з системами запису дзвінків	Низький

На рисунку 2.1 представлено структуру підсистеми аналізу та обробки даних.



Рисунок 2.1 - Структура підсистеми аналізу та обробки даних

Таблиця 2.2 містить детальний перелік функціональних вимог до підсистеми аналізу та обробки даних.

Таблиця 2.2 - Функціональні вимоги до підсистеми аналізу та обробки даних

Код вимоги	Опис вимоги	Пріоритет
FR-DA-01	Обчислення мережевих метрик (затримка, джитер, втрата пакетів)	Високий
FR-DA-02	Обчислення метрик якості голосу (MOS, R-фактор)	Високий
FR-DA-03	Аналіз сигнального трафіку та виявлення проблем	Середній
FR-DA-04	Аналіз медіа-трафіку та виявлення проблем з кодеками	Високий
FR-DA-05	Виявлення аномалій у роботі сервісів IP-телефонії	Високий
FR-DA-06	Кореляція різних симптомів для діагностики проблем	Середній

Функціональні вимоги до підсистеми візуалізації результатів включають створення інтерактивних дашбордів та звітів, систему сповіщень та оповіщень, а також можливості налаштування представлення даних для різних категорій користувачів. Ця підсистема повинна забезпечувати зручний доступ до результатів аналізу та їх візуальне представлення у зрозумілій формі.

У таблиці 2.3 представлено детальний перелік функціональних вимог до підсистеми візуалізації.

Функціональні вимоги до підсистеми адміністрування включають управління користувачами та їх правами, налаштування системи, управління агентами збору даних та забезпечення безпеки. Ця підсистема повинна забезпечувати можливість гнучкого налаштування системи відповідно до

потреб конкретної організації та забезпечувати контроль доступу до даних та функціональності системи.

Таблиця 2.3 - Функціональні вимоги до підсистеми візуалізації

Код вимоги	Опис вимоги	Пріоритет
FR-VZ-01	Створення інтерактивних дашбордів для моніторингу в реальному часі	Високий
FR-VZ-02	Генерація звітів про якість сервісів IP-телефонії	Високий
FR-VZ-03	Система сповіщень про проблеми та аномалії	Високий
FR-VZ-04	Налаштування представлення даних для різних категорій користувачів	Середній
FR-VZ-05	Експорт результатів аналізу в різних форматах (PDF, Excel, CSV)	Середній
FR-VZ-06	Візуалізація тенденцій та прогнозів	Низький
FR-VZ-07	Інтерактивна візуалізація мережевої топології	Середній
FR-VZ-08	Візуалізація маршрутів проходження трафіку	Низький
FR-VZ-09	Налаштування порогових значень для метрик та сповіщень	Високий
FR-VZ-10	Підтримка мобільного доступу до дашбордів та звітів	Низький

На рисунку 2.2 представлено модель потоків даних у системі оцінки сервісів IP-телефонії.

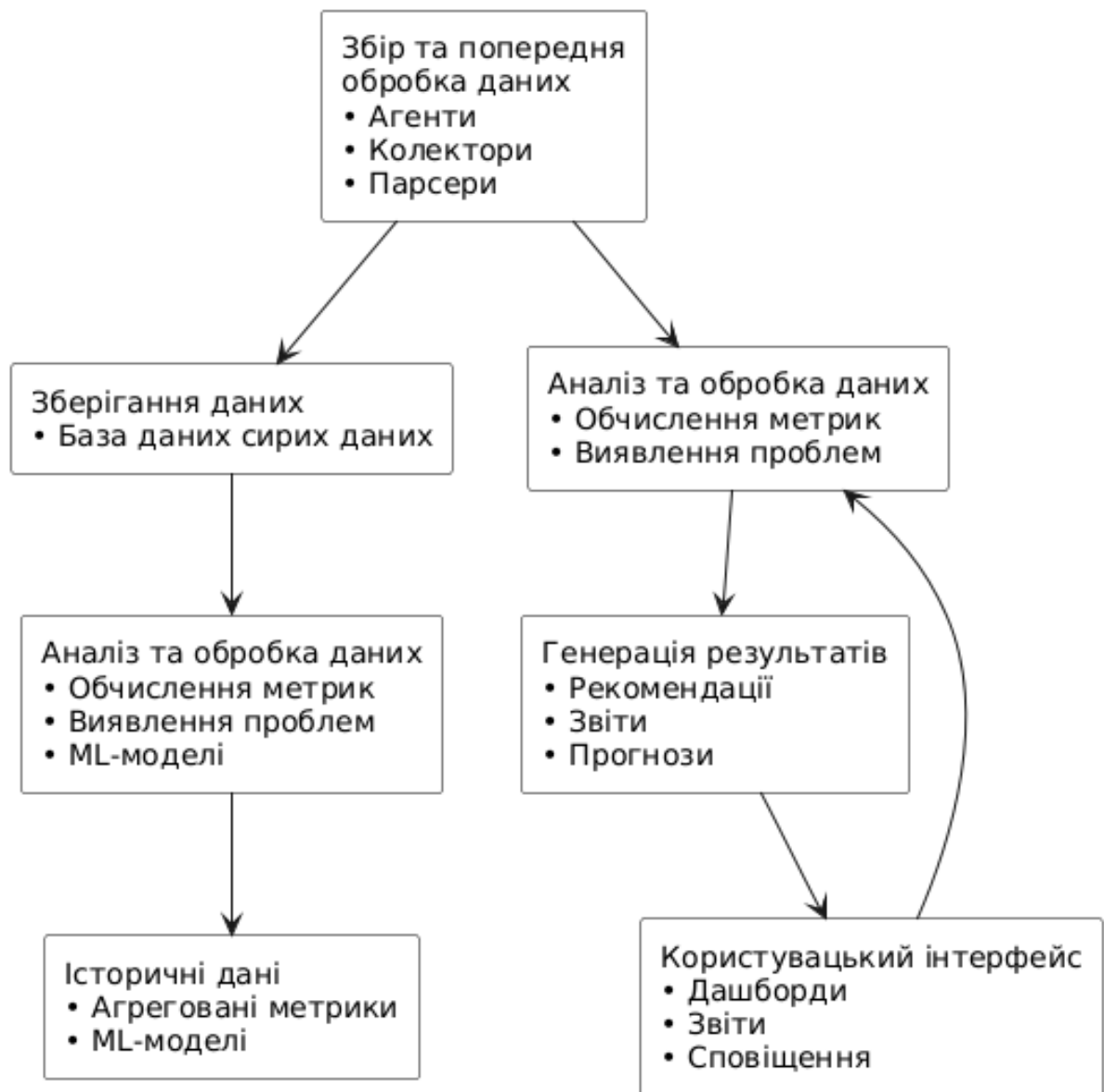


Рисунок 2.2 - Модель потоків даних у системі оцінки сервісів IP-телефонії

Нефункціональні вимоги до програмного забезпечення для оцінки сервісів IP-телефонії включають вимоги до продуктивності, масштабованості, безпеки, надійності, зручності використання та сумісності з іншими системами [3]. Ці вимоги визначають якісні характеристики системи, які необхідно забезпечити при її розробці.

У таблиці 2.4 представлено детальний перелік нефункціональних вимог до системи.

Таблиця 2.4 - Нефункціональні вимоги до програмного забезпечення

Код вимоги	Опис вимоги	Пріоритет
NFR-01	Здатність обробляти дані про не менше 1000 одночасних дзвінків	Високий
NFR-02	Затримка відображення даних в інтерфейсі не більше 2 секунд	Високий
NFR-03	Доступність системи не менше 99.9% часу	Високий
NFR-04	Підтримка горизонтального масштабування для збільшення навантаження	Середній
NFR-05	Захист даних відповідно до вимог GDPR та інших регуляторних норм	Високий
NFR-06	Підтримка автентифікації та авторизації користувачів	Високий
NFR-07	Інтуїтивно зрозумілий інтерфейс, що не вимагає спеціального навчання	Середній
NFR-08	Сумісність з основними браузерами (Chrome, Firefox, Safari, Edge)	Високий
NFR-09	Можливість резервного копіювання та відновлення даних	Середній
NFR-10	Низьке споживання ресурсів агентами збору даних	Середній

Окрім загальних вимог, необхідно враховувати специфічні вимоги різних груп користувачів програмного забезпечення для оцінки сервісів IP-телефонії [4]. Для адміністраторів мереж важливими є детальні технічні метрики та засоби діагностики проблем. IT-менеджери потребують агрегованих метрик та порівняльного аналізу. Керівники зацікавлені в бізнес-метриках та аналізі тенденцій. Оператори контакт-центрів потребують моніторингу в реальному часі та сповіщень про проблеми. На рисунку 2.3

представлено модель варіантів використання системи для різних груп користувачів.



Рисунок 2.3 - Модель варіантів використання системи

На основі сформульованих вимог можна визначити основні компоненти системи та розробити її архітектуру. Система повинна включати наступні основні компоненти: компоненти збору даних (агенти, колектори, парсери); компоненти зберігання даних (бази даних метрик, сховища сирих даних); компоненти аналізу та обробки даних (обчислення метрик, виявлення проблем, машинне навчання); компоненти візуалізації та взаємодії з користувачем (вебінтерфейс, API, система сповіщень).

2.2 Архітектура програмного забезпечення

Після визначення вимог до програмного забезпечення для оцінки сервісів IP-телефонії необхідно розробити архітектуру системи, яка забезпечить ефективне виконання всіх функціональних вимог та відповідатиме нефункціональним вимогам. У цьому підрозділі буде детально описано архітектуру розроблюваного програмного забезпечення, його основні компоненти та взаємодію між ними.

При проєктуванні архітектури програмного забезпечення для оцінки сервісів IP-телефонії були враховані такі принципи: модульність та розділення відповідальності; масштабованість та розподіленість; гнучкість та розширюваність; надійність та відмовостійкість; безпека та захист даних [5]. Ці принципи дозволяють створити систему, яка відповідає сучасним вимогам до програмного забезпечення та може адаптуватися до змін у середовищі експлуатації.

Для забезпечення модульності та гнучкості була обрана мікросервісна архітектура, яка дозволяє розділити систему на незалежні компоненти, що можуть розгортатися та масштабуватися окремо один від одного [6]. Кожен мікросервіс відповідає за певну функцію системи та взаємодіє з іншими сервісами через чітко визначені API. На рисунку 2.4 представлено високорівневу архітектуру системи оцінки сервісів IP-телефонії.

Рівень інтерфейсу відповідає за взаємодію з користувачами системи та зовнішніми системами. Він включає вебінтерфейс, API для інтеграції, дашборди та звіти, а також систему сповіщень. Вебінтерфейс побудований на основі сучасних вебтехнологій (React) та забезпечує доступ до функціональності системи через браузер. API для інтеграції реалізовано за принципами REST та забезпечує можливість взаємодії з системою з боку зовнішніх додатків. Дашборди та звіти базуються на бібліотеках візуалізації даних (D3.js) та дозволяють представити результати аналізу в зручній для користувача формі. Система сповіщень забезпечує оперативне інформування

користувачів про виявлені проблеми та аномалії через різні канали (електронна пошта, SMS, інтеграція з месенджерами та системами управління інцидентами).



Рисунок 2.4 - Високорівнева архітектура системи оцінки сервісів IP-телефонії

Рівень оркестрації відповідає за координацію роботи різних компонентів системи та забезпечення їх взаємодії. Він включає API-шлюз, сервіс автентифікації та авторизації, сервіс конфігурації, реєстр сервісів та систему управління чергами повідомлень. API-шлюз (API Gateway) забезпечує єдину точку входу до системи, маршрутизацію запитів до відповідних сервісів, балансування навантаження та базову фільтрацію запитів. Сервіс автентифікації та авторизації забезпечує управління

користувачами, їх правами та доступом до функціональності системи. Сервіс конфігурації забезпечує централізоване управління налаштуваннями системи та її компонентів. Реєстр сервісів забезпечує динамічне виявлення сервісів та їх взаємодію. Система управління чергами повідомлень (наприклад, RabbitMQ, Apache Kafka) забезпечує асинхронну взаємодію між компонентами системи.

Сервіси збору даних відповідають за збір інформації про роботу сервісів IP-телефонії з різних джерел. Вони включають агенти збору трафіку, колектори даних, інтеграції з API різних систем та засоби для проведення синтетичних тестів. Агенти збору трафіку встановлюються в ключових точках мережі та забезпечують захоплення та первинний аналіз мережевого трафіку, пов'язаного з IP-телефонією. Колектори даних отримують інформацію від агентів та інших джерел, агрегують її та передають на подальшу обробку. Інтеграції з API різних систем (IP-АТС, хмарні сервіси IP-телефонії, системи CDR) забезпечують отримання даних про роботу цих систем через їх програмні інтерфейси. Засоби для проведення синтетичних тестів дозволяють активно перевіряти якість сервісів IP-телефонії через виконання тестових дзвінків та аналіз їх параметрів.

У таблиці 2.5 представлено основні компоненти сервісів збору даних та їх функції.

Сервіси аналізу відповідають за обробку зібраних даних, обчислення метрик якості, виявлення проблем та генерацію рекомендацій. Вони включають сервіс обчислення метрик, сервіс виявлення проблем, сервіс машинного навчання та сервіс генерації рекомендацій. Сервіс обчислення метрик забезпечує розрахунок різних показників якості на основі зібраних даних, включаючи мережеві метрики, метрики якості голосу та метрики надійності.

Таблиця 2.5 - Основні компоненти сервісів збору даних

Компонент	Функції	Технології
Агент збору трафіку	Захоплення та аналіз пакетів, декодування протоколів	libpcap, tshark, gopacket
Колектор даних	Отримання даних від агентів, первинна агрегація	Telegraf, Logstash, Fluentd
Сервіс інтеграції з API	Взаємодія з API різних систем IP-телефонії	RESTful API, SOAP, GraphQL
Сервіс синтетичних тестів	Виконання тестових дзвінків, вимірювання якості	SIPp, WebRTC, різні SDK
Парсер протоколів	Аналіз протоколів сигналізації та медіа-протоколів	Спеціалізовані бібліотеки

Сервіс виявлення проблем забезпечує аналіз обчислених метрик, виявлення аномалій та діагностику проблем. Сервіс машинного навчання забезпечує застосування алгоритмів машинного навчання для класифікації проблем, прогнозування якості та оптимізації параметрів. Сервіс генерації рекомендацій забезпечує формування рекомендацій щодо усунення виявлених проблем та оптимізації конфігурації сервісів IP-телефонії.

На рисунку 2.5 представлено діаграму взаємодії компонентів системи при виявленні та діагностиці проблем з якістю IP-телефонії.

Сервіси зберігання відповідають за зберігання та управління даними системи. Вони включають сховище метрик, сховище конфігурацій, сховище результатів аналізу та довготривале сховище даних. Сховище метрик (наприклад, InfluxDB, Prometheus) забезпечує ефективне зберігання та доступ до часових рядів з метриками якості. Сховище конфігурацій (наприклад, etcd, Consul) забезпечує зберігання та управління налаштуваннями системи та її компонентів. Сховище результатів аналізу (наприклад, MongoDB, Elasticsearch) забезпечує зберігання та доступ до результатів аналізу,

включаючи виявлені проблеми, рекомендації та прогнози. Довготривале сховище даних (наприклад, Apache Hadoop, Amazon S3) забезпечує архівацію та зберігання історичних даних для подальшого аналізу.

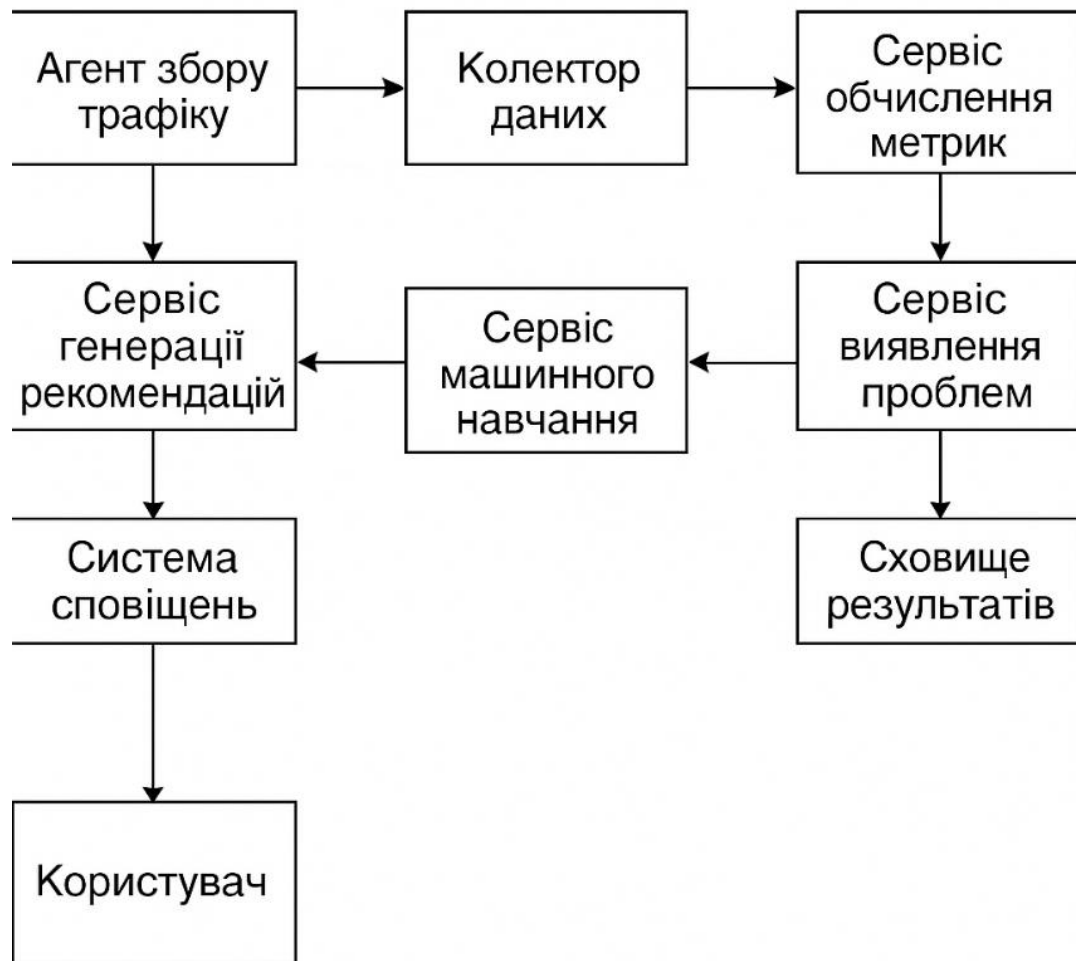


Рисунок 2.5 - Діаграма взаємодії компонентів системи при виявленні та діагностиці проблем

У таблиці 2.6 представлено порівняння різних технологій для реалізації сховища метрик.

Для забезпечення масштабованості та відмовостійкості системи використовуються технології контейнеризації та оркестрації контейнерів, такі як Docker та Kubernetes. Ці технології дозволяють упакувати кожен компонент системи у контейнер, що полегшує розгортання, масштабування

та управління системою. Kubernetes забезпечує автоматичне масштабування, самовідновлення, балансування навантаження та розподілене зберігання стану, що дозволяє системі адаптуватися до змін навантаження та продовжувати роботу навіть при відмові окремих компонентів.

Таблиця 2.6 - Порівняння технологій для реалізації сховища метрик

Технологія	Переваги	Недоліки	Придатність для проєкту
InfluxDB	Висока продуктивність для запису, підтримка SQL-подібної мови запитів, гнучка схема даних	Обмежена масштабованість у безкоштовній версії	Висока
Prometheus	Висока надійність, інтеграція з Kubernetes, потужна система запитів та сповіщень	Складність зберігання довготривалих даних	Середня
TimescaleDB	Сумісність з PostgreSQL, підтримка SQL, висока масштабованість	Вища складність налаштування та управління	Середня
Graphite	Простота використання, ефективне зберігання даних з низькою деталізацією	Обмежена функціональність запитів, складність масштабування	Низька
OpenTSDB	Висока масштабованість, ефективне зберігання даних	Складність налаштування та управління, обмежена функціональність запитів	Низька

Для забезпечення безпеки системи використовуються різні механізми захисту на різних рівнях: мережевий рівень (брандмауери, VPN, ізоляція мереж); рівень автентифікації та авторизації (OAuth 2.0, OIDC, RBAC); рівень передачі даних (TLS/SSL, шифрування даних); рівень додатків (захист від OWASP Top 10, безпечні практики кодування). Ці механізми дозволяють

забезпечити конфіденційність, цілісність та доступність даних та функціональності системи (табл. 2.7).

Таблиця 2.7 - Вибір технологій для реалізації різних рівнів системи

Рівень системи	Обрана технологія	Обґрунтування вибору
Рівень інтерфейсу	React	Висока продуктивність, компонентний підхід, велика спільнота розробників, можливість створення інтерактивних дашбордів
Рівень оркестрації	Kubernetes	Гнучка система оркестрації контейнерів, можливість автоматичного масштабування, самовідновлення та розподіленого зберігання стану
API-шлюз	Kong API Gateway	Відкритий вихідний код, підтримка балансування навантаження, маршрутизації, кешування та автентифікації
Сервіс автентифікації	OAuth 2.0 з Keycloak	Стандартизований протокол автентифікації, підтримка SSO, гнучке управління доступом
Сервіси збору даних	Go (для агентів), Node.js (для колекторів)	Go забезпечує високу продуктивність і низьке споживання ресурсів для агентів, Node.js - гнучкість для інтеграцій
Сервіси аналізу	Python з TensorFlow і scikit-learn	Потужні бібліотеки для машинного навчання та аналізу даних, простота реалізації алгоритмів
Сховище метрик	InfluxDB	Висока продуктивність для запису часових рядів, підтримка SQL-подібної мови запитів, гнучка схема даних
Сховище конфігурацій	etcd	Розподілене сховище ключ-значення з підтримкою консенсусу, інтеграція з Kubernetes
Сховище результатів аналізу	MongoDB	Гнучка схема даних, підтримка складних запитів, масштабованість
Система черг повідомлень	Apache Kafka	Висока пропускна здатність, підтримка потокової обробки, розподіленість
Система візуалізації	Grafana	Розширені можливості для створення дашбордів, підтримка різних джерел даних, система сповіщень
CI/CD pipeline	GitLab CI	Інтеграція з системою контролю версій, гнучкі налаштування, підтримка контейнеризації

Для реалізації модуля машинного навчання використовуються сучасні бібліотеки та фреймворки, такі як TensorFlow, PyTorch, scikit-learn, що дозволяють ефективно реалізувати різні алгоритми машинного навчання та глибокого навчання. Модуль включає компоненти для збору та підготовки

даних, навчання моделей, використання моделей та моніторингу їх ефективності. Це дозволяє постійно покращувати якість прогнозування та рекомендацій на основі накопичених даних.

Особливу увагу при проектуванні архітектури було приділено питанням інтеграції з існуючими системами моніторингу та управління ІТ-інфраструктурою. Система надає API для інтеграції з системами управління інцидентами (ServiceNow, JIRA), системами моніторингу (Zabbix, Nagios), системами ITSM (Remedy, ServiceNow) та іншими системами. Це дозволяє вбудувати оцінку якості сервісів IP-телефонії в загальні процеси управління ІТ-інфраструктурою організації.

Для управління розробкою та розгортанням програмного забезпечення використовуються практики DevOps, такі як безперервна інтеграція (CI) та безперервне розгортання (CD). Ці практики дозволяють автоматизувати процеси збірки, тестування та розгортання системи, що зменшує час виведення нових функцій на ринок та підвищує якість програмного забезпечення. Для реалізації CI/CD використовуються такі інструменти, як Jenkins, GitLab CI, GitHub Actions.

2.3 Алгоритми та методи оцінки сервісів IP-телефонії

Алгоритми оцінки сервісів IP-телефонії можна розділити на кілька груп залежно від їх призначення та характеру оцінюваних параметрів: алгоритми оцінки мережевих параметрів; алгоритми оцінки якості голосу; алгоритми виявлення аномалій та діагностики проблем; алгоритми прогнозування якості та оптимізації параметрів [2]. Розглянемо кожен групу алгоритмів детальніше.

Алгоритми оцінки мережевих параметрів призначені для вимірювання та аналізу таких характеристик мережі, як затримка, джитер, втрата пакетів та пропускна здатність. Ці параметри безпосередньо впливають на якість передачі голосу через IP-мережі. Для вимірювання затримки

використовується метод Round-Trip Time (RTT), який передбачає відправлення пакетів до адресата та вимірювання часу, необхідного для отримання відповіді [4]. Для оцінки джитера (варіації затримки) використовується алгоритм, який обчислює стандартне відхилення часу прибуття пакетів. На рисунку 2.6 представлено алгоритм обчислення джитера на основі часу прибуття пакетів.

```

1. Ініціалізувати масив timestamp[] - час прибуття пакетів
2. Ініціалізувати масив delta[] - різниця між часом прибуття сусідніх пакетів
3. Для кожного пакета i від 1 до n-1:
   3.1. delta[i] = timestamp[i] - timestamp[i-1]
4. Обчислити середнє значення масиву delta[]:
   4.1. avg_delta = сума(delta) / (n-1)
5. Обчислити суму квадратів відхилень:
   5.1. sum_sq_diff = 0
   5.2. Для кожного i від 0 до n-2:
       5.2.1. diff = delta[i] - avg_delta
       5.2.2. sum_sq_diff = sum_sq_diff + diff^2
6. Обчислити джитер як стандартне відхилення:
   6.1. jitter = sqrt(sum_sq_diff / (n-1))
7. Повернути значення jitter

```

Рисунок 2.6 - Алгоритм обчислення джитера на основі часу прибуття пакетів

Для оцінки втрати пакетів використовується алгоритм, який порівнює кількість відправлених та отриманих пакетів. У RTP-протоколі кожен пакет має порядковий номер, що дозволяє виявляти втрачені пакети. Формула (2.1) для обчислення відсотка втрати пакетів наведена нижче:

$$PacketLoss = (lost_packets / total_packets) \times 100\%, \quad (2.1)$$

де *lost_packets* - кількість втрачених пакетів, *total_packets* - загальна кількість відправлених пакетів.

Для оцінки якості голосу в IP-телефонії використовуються різні методи, серед яких найбільш поширеними є E-модель, PESQ та POLQA [4]. E-модель (ITU-T G.107) дозволяє обчислити R-фактор, який потім може бути перетворений в оцінку MOS (Mean Opinion Score). R-фактор обчислюється за формулою (2.2):

$$R = R0 - Is - Id - Ie + A, \quad (2.2)$$

де $R0$ - базовий рівень якості, Is - погіршення, пов'язане з сигналом, Id - погіршення, пов'язане з затримкою, Ie - погіршення, пов'язане з використанням кодека та втратою пакетів, A - фактор очікування (компенсація).

Таблиця 2.7 містить значення параметрів Е-моделі для різних кодеків за стандартом ITU-T G.113.

Таблиця 2.7 - Значення параметрів Е-моделі для різних кодеків

Кодек	Бітрейт (kbps)	Ie (без втрат)	Bpl (стійкість до втрат)
G.711	64	0	4.3
G.729	8	10	19
G.723.1	6.3	15	15
G.723.1	5.3	19	14
G.726	32	7	23
G.726	24	16	23
G.728	16	7	20
GSM-FR	13	20	10
GSM-HR	5.6	23	9
GSM-EFR	12.2	5	10

Для обчислення погіршення Ie з урахуванням втрати пакетів використовується формула (2.3):

$$Ie_{eff} = Ie + (95 - Ie) * (Ppl / (Ppl + Bpl)), \quad (2.3)$$

де I_e - погіршення, пов'язане з використанням кодека без втрати пакетів, Ppl - відсоток втрати пакетів, Bpl - стійкість кодека до втрати пакетів.

Після обчислення R -фактора, його можна перетворити в оцінку MOS за формулою (2.4):

$$\text{If } R < 0: \quad MOS = 1.0$$

$$\text{If } 0 < R < 100: MOS = 1 + 0.035 \times R + R \times (R - 60) \times (100 - R) \times 7 \times 10^{-6} \quad (2.4)$$

$$\text{If } R > 100: \quad MOS = 4.5.$$

PESQ (Perceptual Evaluation of Speech Quality, ITU-T P.862) та POLQA (Perceptual Objective Listening Quality Analysis, ITU-T P.863) є більш складними методами, які порівнюють оригінальний та отриманий голосові сигнали та обчислюють оцінку якості на основі психоакустичних моделей сприйняття мовлення людиною [5]. Ці методи дають більш точні результати, але вимагають доступу до оригінального сигналу, що не завжди можливо в реальних умовах експлуатації.

У розроблюваному програмному забезпеченні реалізовано алгоритм обчислення оцінки MOS на основі E-моделі, який використовує мережеві параметри (затримка, джитер, втрата пакетів) та інформацію про використовуваний кодек. Цей алгоритм дозволяє оцінювати якість голосу в реальному часі без необхідності порівняння з оригінальним сигналом. На рисунку 2.7 представлено блок-схему алгоритму обчислення оцінки MOS.

Алгоритми виявлення аномалій та діагностики проблем призначені для автоматичного виявлення нетипової поведінки системи та визначення можливих причин проблем з якістю [6]. У розроблюваному програмному забезпеченні реалізовано кілька підходів до виявлення аномалій: статистичний аналіз часових рядів; методи машинного навчання; правила на основі експертних знань.

Статистичний аналіз часових рядів базується на обчисленні статистичних характеристик метрик (середнє значення, стандартне

відхилення, персентилі) та виявленні відхилень від нормальної поведінки. Наприклад, значення метрики може вважатися аномальним, якщо воно перевищує середнє значення більш ніж на три стандартні відхилення.

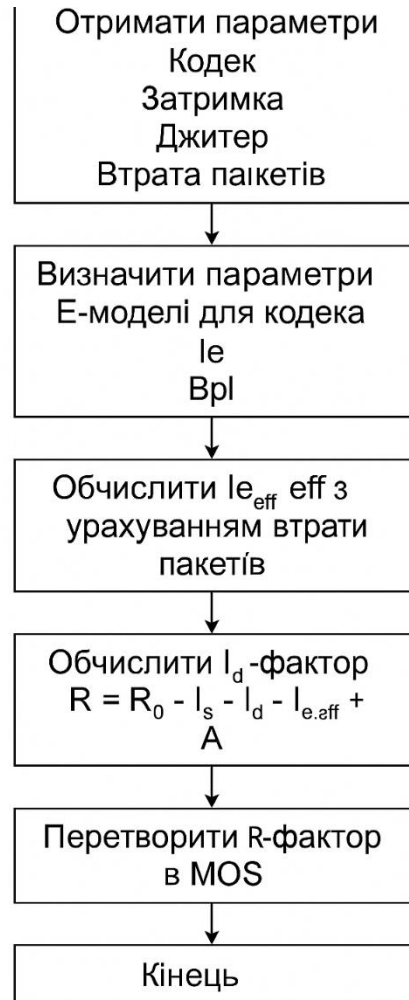


Рисунок 2.7 - Блок-схема алгоритму обчислення оцінки MOS на основі Е-моделі

Методи машинного навчання для виявлення аномалій включають алгоритми кластеризації (наприклад, DBSCAN, K-means), алгоритми класифікації (наприклад, Random Forest, Support Vector Machines) та спеціалізовані алгоритми для виявлення аномалій (наприклад, Isolation Forest, One-Class SVM) [4]. Ці методи дозволяють виявляти складні патерни в даних та автоматично адаптуватися до змін у характеристиках мережі та трафіку.

Правила на основі експертних знань базуються на встановлених порогових значеннях для різних метрик та логічних умовах, які визначають аномальну поведінку. Наприклад, правило може визначати, що комбінація високої затримки, високого джитера та високого рівня втрати пакетів вказує на проблеми з перевантаженням мережі. У таблиці 2.8 представлено приклади правил для діагностики проблем з якістю IP-телефонії.

Таблиця 2.8 - Приклади правил для діагностики проблем з якістю IP-телефонії

ID правила	Умова	Діагноз	Рекомендації
R001	Висока затримка (>150 мс) AND Низький джитер AND Низька втрата пакетів	Проблеми з маршрутизацією	Перевірити маршрутизацію, використати альтернативні маршрути
R002	Низька затримка AND Високий джитер (>30 мс) AND Низька втрата пакетів	Проблеми з буферизацією	Збільшити розмір джитер-буфера, налаштувати QoS
R003	Низька затримка AND Низький джитер AND Висока втрата пакетів (>1%)	Проблеми з перевантаженням на окремих ділянках	Збільшити пропускну здатність, налаштувати QoS
R004	Висока затримка AND Високий джитер AND Висока втрата пакетів	Серйозні проблеми з мережею	Повністю перевірити мережу, виявити вузькі місця
R005	Різка зміна метрик у певний час доби	Проблеми з пропускну здатністю у години пік	Збільшити пропускну здатність, розподілити навантаження

У розроблюваному програмному забезпеченні реалізовано гібридний підхід до виявлення аномалій та діагностики проблем, який поєднує статистичний аналіз, методи машинного навчання та правила на основі

експертних знань. Це дозволяє ефективно виявляти різні типи проблем та забезпечувати високу точність діагностики.

Алгоритми прогнозування якості та оптимізації параметрів призначені для передбачення майбутньої якості сервісів IP-телефонії та визначення оптимальних налаштувань для забезпечення найкращої якості [8]. У розроблюваному програмному забезпеченні реалізовано алгоритми прогнозування на основі часових рядів (ARIMA, SARIMA) та методів машинного навчання (регресія, нейронні мережі). Ці алгоритми дозволяють прогнозувати зміни в якості сервісу на основі історичних даних та поточних тенденцій.

Для оптимізації параметрів використовуються методи оптимізації, такі як генетичні алгоритми, градієнтний спуск та байєсівська оптимізація. Ці методи дозволяють знайти оптимальні значення параметрів, які забезпечують найкращу якість сервісу в заданих умовах. Наприклад, можна оптимізувати параметри QoS (Quality of Service) для забезпечення найкращої якості голосу при обмеженій пропускній здатності мережі.

У лістингу 2.1 представлено псевдокод алгоритму для прогнозування якості сервісів IP-телефонії на основі методу SARIMA (Seasonal AutoRegressive Integrated Moving Average).

Лістинг 2.1 - Псевдокод алгоритму прогнозування якості на основі SARIMA

```
# Функція для прогнозування якості сервісів IP-телефонії
def predict_quality(historical_data, forecast_horizon):
    # Підготовка даних
    data = preprocess_data(historical_data)

    # Визначення оптимальних параметрів моделі SARIMA
    # p, d, q - несезонні параметри
    # P, D, Q, s - сезонні параметри
    best_params = find_optimal_sarima_params(data)
    p, d, q, P, D, Q, s = best_params
```

```

# Створення та навчання моделі SARIMA
model = SARIMA(p, d, q, P, D, Q, s)
model.fit(data)

# Прогнозування майбутніх значень
forecast = model.predict(forecast_horizon)

# Обчислення довірчих інтервалів
confidence_intervals = calculate_confidence_intervals(model,
forecast_horizon)

# Повернення результатів прогнозування
return {
    'forecast': forecast,
    'confidence_intervals': confidence_intervals,
    'model_params': best_params
}

# Функція для визначення оптимальних параметрів моделі SARIMA
def find_optimal_sarima_params(data):
    # Ініціалізація змінних для зберігання найкращих параметрів
    best_aic = float('inf')
    best_params = None

    # Діапазони параметрів для пошуку
    p_range = range(0, 3)
    d_range = range(0, 2)
    q_range = range(0, 3)
    P_range = range(0, 2)
    D_range = range(0, 2)
    Q_range = range(0, 2)
    s_values = [24, 168] # Сезонності: добова (24 години) та
тижнева (168 годин)

    # Перебір всіх комбінацій параметрів
    for p in p_range:
        for d in d_range:
            for q in q_range:

```

```

for P in P_range:
    for D in D_range:
        for Q in Q_range:
            for s in s_values:
                try:
                    # Створення та навчання
                    моделі з поточними параметрами
                    model = SARIMA(p, d, q, P,
                    D, Q, s)
                    model_fit = model.fit(data)

                    # Обчислення критерію AIC
                    aic = model_fit.aic

                    # Оновлення найкращих
                    параметрів, якщо поточний AIC кращий
                    if aic < best_aic:
                        best_aic = aic
                        best_params = (p, d, q,
                        P, D, Q, s)
                except:
                    # Ігнорування помилок при
                    невдалому навчанні моделі
                    continue

            # Повернення найкращих параметрів
            return best_params

```

Для обчислення комплексної оцінки якості сервісів IP-телефонії використовується інтегральний показник, який враховує різні аспекти якості [5]. Цей показник обчислюється за формулою (2.5):

$$QoS_index = w1 \times NQ + w2 \times VQ + w3 \times R + w4 \times F \quad (2.5)$$

де NQ - оцінка мережевих параметрів, VQ - оцінка якості голосу, R - оцінка надійності, F - оцінка функціональності, $w1$, $w2$, $w3$, $w4$ - вагові коефіцієнти, що визначають важливість різних аспектів якості.

Вагові коефіцієнти можуть бути налаштовані відповідно до пріоритетів конкретної організації. Наприклад, для контакт-центрів критично важливою є якість голосу, тому ваговий коефіцієнт $w2$ може бути встановлений більшим, ніж інші коефіцієнти. У таблиці 2.9 представлено приклади різних вагових схем для різних типів організацій.

Таблиця 2.9 - Приклади вагових схем для обчислення інтегрального показника якості

Тип організації	$w1$ (мережеві параметри)	$w2$ (якість голосу)	$w3$ (надійність)	$w4$ (функціональність)
Контакт-центр	0.2	0.4	0.3	0.1
Фінансова установа	0.2	0.3	0.4	0.1
ІТ-компанія	0.2	0.2	0.2	0.4
Державна установа	0.3	0.3	0.3	0.1
Медичний заклад	0.2	0.3	0.4	0.1

2.4 Висновки за розділом 2

У другому розділі було проведено проектування програмного забезпечення для оцінки сервісів ІР-телефонії, що дозволило визначити основні вимоги, архітектурні рішення та алгоритми оцінки якості.

Під час формулювання вимог до програмного забезпечення було виділено функціональні вимоги до підсистем збору даних, аналізу й обробки

даних та візуалізації результатів. Також були визначені нефункціональні вимоги, що включають параметри продуктивності, масштабованості, безпеки та зручності використання. Особливу увагу було приділено забезпеченню можливості обробки даних не менше 1000 одночасних дзвінків та доступності системи не менше 99.9% часу.

Для забезпечення всіх сформульованих вимог була розроблена мікросервісна архітектура системи, що складається з чотирьох основних рівнів: інтерфейсу, оркестрації, сервісів (збору даних, аналізу, зберігання) та сховищ даних. Такий підхід дозволяє забезпечити модульність, масштабованість та гнучкість системи, а також можливість незалежного розгортання та оновлення окремих компонентів.

У розділі також було розроблено алгоритми оцінки якості сервісів IP-телефонії, включаючи алгоритми обчислення мережевих параметрів (затримка, джитер, втрата пакетів), оцінки якості голосу на основі E-моделі, виявлення аномалій та прогнозування якості.

3 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Структура бази даних

Для зберігання часових рядів метрик якості IP-телефонії використовується InfluxDB – спеціалізована база даних, що забезпечує високу продуктивність запису та ефективне зберігання великих обсягів метрик. Організація даних в InfluxDB ґрунтується на концепціях вимірювань (measurements), що є аналогом таблиць у реляційних базах даних; тегів (tags) – пар ключ-значення для індексації та швидкого пошуку; полів (fields) – пар ключ-значення, що містять фактичні метрики; та часових міток (timestamp), що вказують час запису даних. На рисунку 3.1 представлено концептуальну модель організації даних у сховищі метрик InfluxDB.

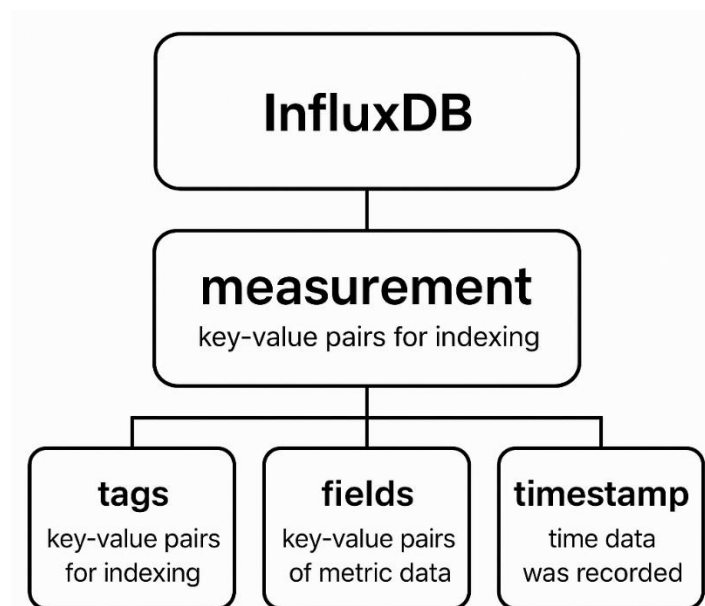


Рисунок 3.1 - Концептуальна модель організації даних у сховищі метрик InfluxDB

Для програмного забезпечення оцінки сервісів IP-телефонії розроблено ряд основних вимірювань (measurements) для зберігання метрик, включаючи: voip_network_metrics для мережеских метрик IP-телефонії (затримка, джитер, втрата пакетів); voip_call_quality для метрик якості голосу

(MOS, R-фактор); `voip_signal_metrics` для метрик сигнального трафіку; `voip_service_availability` для метрик доступності сервісів; та `voip_synthetic_tests` для результатів синтетичних тестів. У таблиці 3.1 представлено структуру вимірювання `voip_network_metrics` для зберігання мережевих метрик IP-телефонії.

Таблиця 3.1 - Структура вимірювання `voip_network_metrics` у InfluxDB

Категорія	Назва	Тип	Опис
Tags	<code>service_id</code>	string	Ідентифікатор сервісу IP-телефонії
Tags	<code>source_ip</code>	string	IP-адреса джерела
Tags	<code>destination_ip</code>	string	IP-адреса призначення
Tags	<code>codec</code>	string	Використовуваний кодек
Tags	<code>location</code>	string	Географічна локація
Tags	<code>agent_id</code>	string	Ідентифікатор агента збору даних
Fields	<code>latency</code>	float	Затримка в мілісекундах
Fields	<code>jitter</code>	float	Джитер в мілісекундах
Fields	<code>packet_loss</code>	float	Відсоток втрати пакетів
Fields	<code>bandwidth</code>	float	Використана пропускна здатність (kbps)
Fields	<code>mos_estimate</code>	float	Оцінка MOS на основі мережевих параметрів
Fields	<code>r_factor</code>	float	R-фактор за E-моделлю

Для зберігання результатів аналізу та виявлених проблем використовується MongoDB – документо-орієнтована база даних, що забезпечує гнучку схему даних та підтримку складних запитів. Структура колекцій у MongoDB розроблена для ефективного зберігання інформації про виявлені проблеми, їх діагностику та рекомендації щодо усунення. На рисунку 3.2 представлено схему основних колекцій в MongoDB та зв'язки між ними.

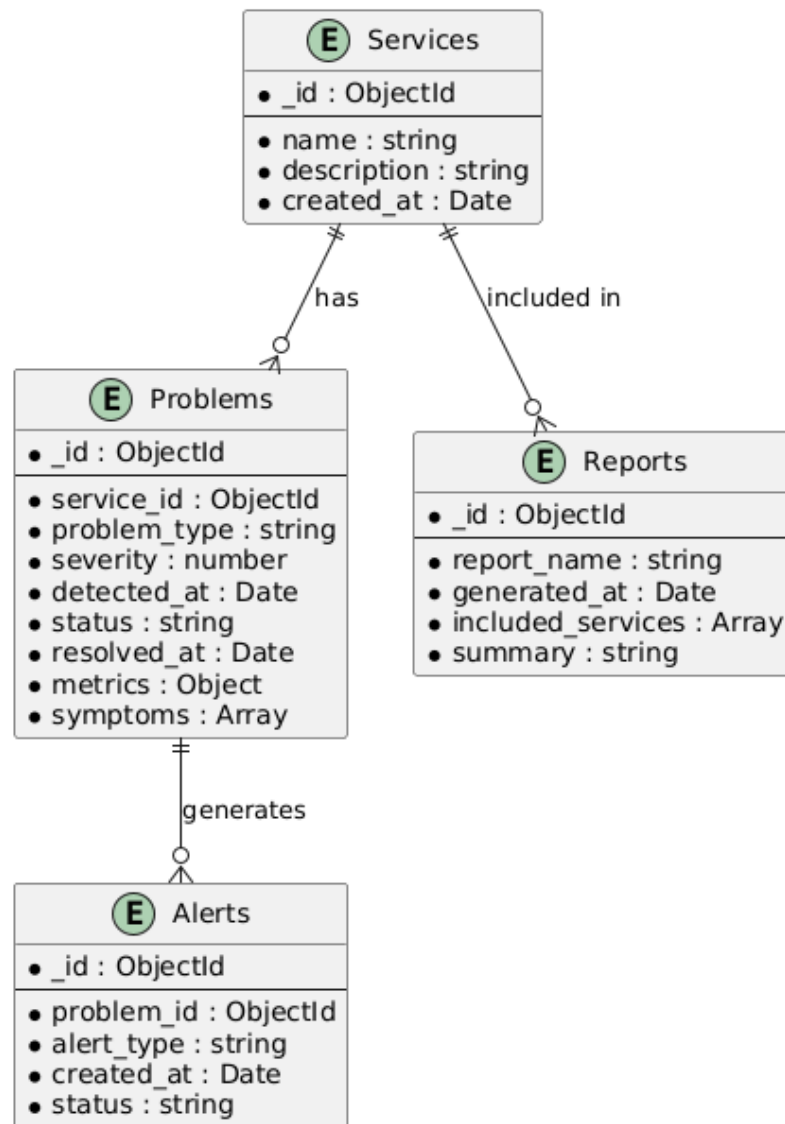


Рисунок 3.2 - Схема колекцій бази даних MongoDB для зберігання результатів аналізу

Основні колекції в MongoDB включають: **Services** для зберігання інформації про сервіси IP-телефонії; **Problems** для документування виявлених проблем та аномалій; **Alerts** для реєстрації створених сповіщень; та **Reports** для зберігання згенерованих звітів. Для імплементції зв'язків між документами використовується підхід з вбудованими документами та зовнішніми посиланнями, що дозволяє оптимізувати запити відповідно до типових сценаріїв використання системи. Структура документа в колекції **Problems** представлена в таблиці 3.2.

Таблиця 3.2 - Структура документа в колекції Problems MongoDB

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор проблеми
service_id	string	Ідентифікатор сервісу IP-телефонії
problem_type	string	Тип проблеми (network, codec, signal, etc.)
severity	number	Рівень серйозності (1-5)
detected_at	Date	Час виявлення проблеми
status	string	Статус проблеми (new, acknowledged, resolved)
resolved_at	Date	Час розв'язання проблеми
metrics	Object	Метрики, пов'язані з проблемою
symptoms	Array	Симптоми проблеми

Для зберігання конфігурацій системи та її компонентів використовується etcd – розподілене сховище ключ-значення, яке забезпечує високу доступність та узгодженість даних. Структура ключів в etcd організована ієрархічно, що дозволяє логічно групувати конфігураційні параметри та ефективно керувати доступом до них. Основні категорії конфігураційних параметрів включають: налаштування системи моніторингу; порогові значення для метрик та сповіщень; конфігурацію агентів збору даних; параметри алгоритмів аналізу та машинного навчання.

Відповідно до алгоритму обчислення оцінки MOS, описаного в розділі 2.3 (рис. 2.7), дані з мережевих метрик проходять кілька етапів обробки перед збереженням у базі даних. Спочатку агенти збору даних фіксують базові мережеві параметри (затримка, джитер, втрата пакетів) та передають їх через Kafka у топик raw_metrics. Далі сервіс обчислення метрик отримує ці дані, застосовує алгоритм Е-моделі для обчислення R-фактора та MOS, і записує результати в InfluxDB. Процес збереження даних та їх проходження через систему представлено на рисунку 3.3.



Рисунок 3.3 - Діаграма процесів обробки та зберігання даних в системі оцінки сервісів IP-телефонії

Для забезпечення ефективного доступу до даних та їх аналізу розроблено набір запитів та індексів, оптимізованих для типових сценаріїв використання системи. В InfluxDB створено індекси за основними тегами (`service_id`, `location`, `codec`), що дозволяє швидко отримувати дані для конкретних сервісів та географічних локацій. У MongoDB створено індекси за полями `service_id`, `problem_type`, `severity` та `status` для оптимізації запитів до колекції `Problems`.

Для реалізації моделі прогнозування якості на основі алгоритму SARIMA, описаного в лістингу 2.2, дані з InfluxDB агрегуються за часовими інтервалами (годинні, добові, тижневі) та передаються в сервіс машинного навчання для побудови прогнозних моделей. Результати прогнозування зберігаються в окремому вимірюванні `voip_quality_forecasts` в InfluxDB, що дозволяє ефективно візуалізувати прогнози разом з фактичними даними.

3.2 Розробка інтерфейсу програмного забезпечення

Інтерфейс користувача є ключовим елементом програмного забезпечення для оцінки сервісів IP-телефонії, оскільки від його якості наряду залежить ефективність роботи користувачів різних категорій, від адміністраторів мереж до керівників IT-відділів. Відповідно до архітектурних рішень, прийнятих у розділі 2, для розробки інтерфейсу було обрано технологію React з Redux, що забезпечує високу продуктивність та гнучкість при створенні інтерактивних вебдодатків.

При розробці інтерфейсу було враховано принципи User-Centered Design (UCD), зокрема фокус на потребах різних груп користувачів, простоту використання та інформативність представлення даних. Основна структура інтерфейсу користувача представлена на рисунку 3.4.

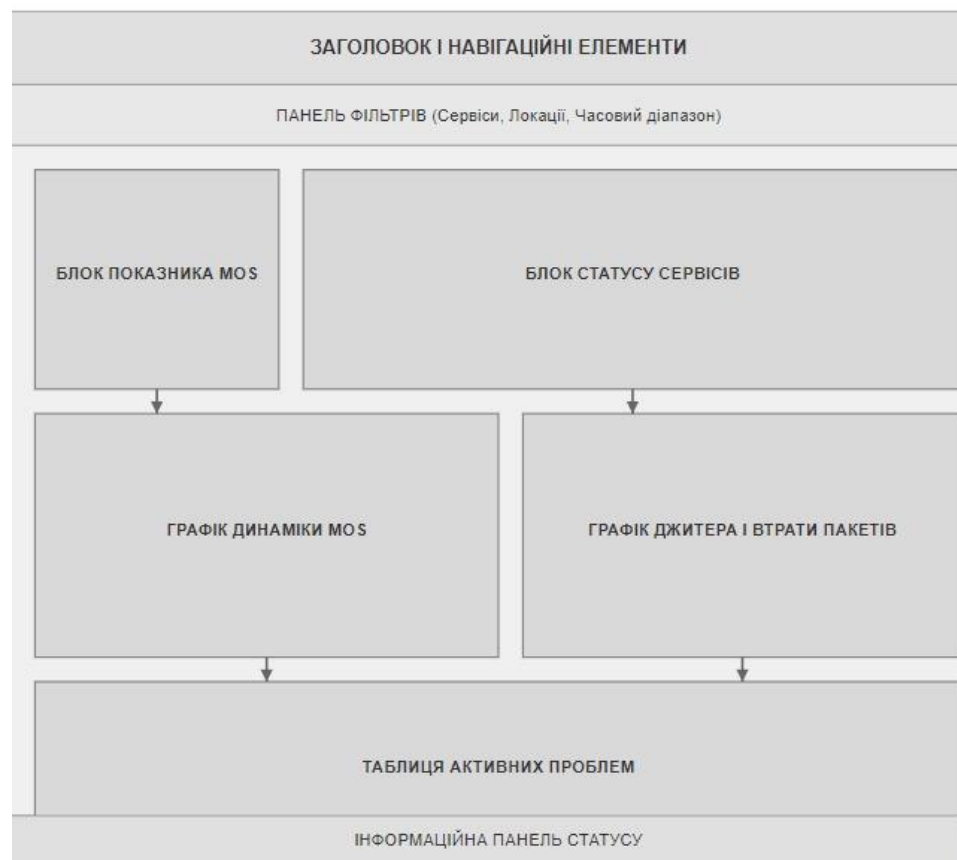


Рисунок 3.4 - Загальна структура інтерфейсу користувача програмного забезпечення

Інтерфейс програмного забезпечення для оцінки сервісів ІР-телефонії поділено на кілька функціональних зон: навігаційне меню, що забезпечує доступ до різних модулів системи; основна робоча область для відображення даних та взаємодії з ними; панель фільтрів та управління для налаштування відображуваних даних; зона сповіщень для відображення важливих повідомлень; та панель статусу для показу загального стану системи.

У таблиці 3.4 представлено основні теми та палітри кольорів, використані в інтерфейсі.

Таблиця 3.4 - Основні теми та палітри кольорів інтерфейсу

Елемент інтерфейсу	Основний колір	Додатковий колір	Примітка
Заголовок та навігація	#1976d2	#ffffff	Темно-синій з білим текстом
Кнопки основні	#2196f3	#ffffff	Синій з білим текстом
Фон основний	#f5f5f5	#000000	Світло-сірий з чорним текстом
Картки та панелі	#ffffff	#000000	Білий з чорним текстом
Статус нормальний	#4caf50	#ffffff	Зелений з білим текстом
Статус попередження	#ff9800	#000000	Помаранчевий з чорним текстом
Статус критичний	#f44336	#ffffff	Червоний з білим текстом
Графіки, ряд 1	#2196f3	-	Синій
Графіки, ряд 2	#4caf50	-	Зелений
Графіки, ряд 3	#ff9800	-	Помаранчевий

Центральним елементом інтерфейсу є дашборди – інтерактивні панелі для візуалізації ключових метрик та стану сервісів IP-телефонії.

Для реалізації інтерактивних графіків використано бібліотеку D3.js, що дозволяє створювати складні візуалізації з можливістю масштабування, панорамування та деталізації даних. Архітектура компонентів візуалізації розроблена з урахуванням принципів повторного використання та композиції, що дозволяє гнучко комбінувати різні типи візуалізацій відповідно до потреб користувачів.

Для забезпечення адаптивності інтерфейсу та його коректного відображення на різних пристроях (десктоп, планшет, смартфон) використано підхід Responsive Web Design з застосуванням CSS Grid Layout та Flexbox. Це дозволяє динамічно адаптувати розташування елементів інтерфейсу залежно від розміру екрану та орієнтації пристрою. На рисунку 3.7 представлено приклад адаптивного дашборду на різних пристроях.

3.3 Визначення порядку роботи програми

Розроблене програмне забезпечення для оцінки сервісів IP-телефонії складається з взаємопов'язаних компонентів, реалізованих за мікросервісною архітектурою. Розглянемо структуру проєкту та основні програмні модулі, які забезпечують функціонування системи.

Кореневий каталог проєкту містить конфігураційні файли для розгортання системи та управління залежностями. Файл `docker-compose.yml` визначає конфігурацію контейнерів для локальної розробки, а директорія `kubernetes/` включає маніфести для розгортання в кластері Kubernetes. Файл `package.json` у кореновому каталозі містить скрипти для управління життєвим циклом розробки та залежності рівня проєкту. Структура основних директорій проєкту представлена на рисунку 3.5.

```

Проект (корінь)
├── docker-compose.yml
├── package.json
├── .gitlab-ci.yml
├── .env.production
├── .env.development
├── kubernetes/
│   └── charts/
├── backend/
│   ├── api-gateway/
│   │   ├── server.js
│   │   ├── routes/
│   │   │   └── index.js
│   │   └── middleware/
│   ├── collector-service/
│   │   ├── collector.js
│   │   ├── config.js
│   │   └── lib/
│   │       ├── pcap-parser.js
│   │       ├── sip-analyzer.js
│   │       └── rtcp-analyzer.js
│   ├── analysis-service/
│   │   ├── app.py
│   │   └── analyzers/
│   │       ├── mos_calculator.py
│   │       └── anomaly_detector.py
│   ├── storage-service/
│   │   ├── influx-client.js
│   │   ├── mongo-client.js
│   │   └── schemas/
│   ├── agents/
│   │   ├── main.go
│   │   └── collectors/
│   └── frontend/
│       └── src/
│           ├── components/
│           │   ├── dashboard/
│           │   │   ├── MetricsGrid.jsx
│           │   │   ├── MetricCard.jsx
│           │   │   └── Chart.jsx
│           │   └── analysis/
│           │       ├── ProblemsList.jsx
│           │       ├── ProblemDetails.jsx
│           │       └── ServiceComparison.jsx
│           ├── api/
│           │   ├── api.js
│           │   └── auth.js
│           ├── redux/
│           └── __tests__/_

```

Рисунок 3.5 - Структура директорій проекту

Серверна частина програмного забезпечення організована в директорії `backend/`, яка включає кілька мікросервісів, кожен у своїй піддиректорії. Директорія `backend/api-gateway/` містить реалізацію API-шлюзу на базі Node.js з використанням Express.js. Основний файл `server.js` ініціалізує сервер, налаштовує маршрутизацію та проміжне програмне забезпечення. Файл `routes/index.js` визначає доступні API-ендпоінти, а директорія `middleware/` містить модулі для автентифікації, авторизації та обробки помилок. API-шлюз забезпечує єдину точку входу для всіх запитів клієнтів та їх маршрутизацію до відповідних мікросервісів.

Директорія `backend/collector-service/` містить мікросервіс для збору та первинної обробки даних про мережеві параметри IP-телефонії. Основний файл `collector.js` реалізує логіку отримання даних від агентів, їх валідації та відправки до сервісу аналізу через Kafka. Модуль `lib/pcap-parser.js` забезпечує аналіз мережевих пакетів для отримання метрик якості, а `lib/sip-analyzer.js` виконує парсинг SIP-трафіку. Таблиця 3.5 описує основні модулі сервісу збору даних.

Таблиця 3.5 - Основні модулі сервісу збору даних

Модуль	Призначення	Технології
<code>collector.js</code>	Отримання та відправка даних	Node.js, Kafka
<code>pcap-parser.js</code>	Аналіз мережевих пакетів	<code>libpcap</code> , Node.js
<code>sip-analyzer.js</code>	Парсинг SIP-трафіку	Node.js
<code>rtcp-analyzer.js</code>	Аналіз RTCP-пакетів	Node.js
<code>config.js</code>	Конфігурація сервісу	Node.js

Директорія `backend/analysis-service/` реалізує мікросервіс аналізу даних на Python з використанням фреймворків Flask та `scikit-learn`. Основний файл `app.py` налаштовує вебсервер та API для взаємодії з іншими компонентами системи. Модуль `analyzers/mos_calculator.py` реалізує алгоритм обчислення оцінки MOS на основі E-моделі, а `analyzers/anomaly_detector.py` використовує алгоритми машинного навчання для виявлення аномалій у метриках якості.

Директорія `backend/storage-service/` реалізує мікросервіс зберігання даних, який відповідає за взаємодію з базами даних InfluxDB та MongoDB. Модуль `influx-client.js` забезпечує збереження метрик у InfluxDB, а `mongo-client.js` - збереження інформації про проблеми та результати аналізу в MongoDB. Файл `schemas/` містить схеми даних для MongoDB.

Директорія `backend/agents/` містить код для агентів збору даних, які встановлюються на різних ділянках мережі для моніторингу параметрів IP-телефонії. Агенти розроблені на мові Go для забезпечення високої продуктивності та низького споживання ресурсів. Основний файл `main.go` ініціалізує агент, а директорія `collectors/` містить модулі для збору різних типів метрик.

Клієнтська частина програмного забезпечення розміщена в директорії `frontend/` і реалізована з використанням React та Redux. Основна структура директорії відповідає стандартній структурі React-додатку, створеного за допомогою Create React App. Директорія `src/components/` містить React-компоненти, організовані за функціональним призначенням, а `src/redux/` - модулі для управління станом додатку. Для стилізації компонентів використовується CSS-in-JS бібліотека `Styled Components`.

Директорія `frontend/src/components/dashboard/` містить компоненти для дашбордів моніторингу, які відображають метрики якості в реальному часі. Компонент `MetricsGrid.jsx` організовує відображення метрик у вигляді сітки, а компоненти `MetricCard.jsx` і `Chart.jsx` відповідають за відображення окремих метрик та їх візуалізацію. Для побудови графіків використовується бібліотека `D3.js`, інтегрована з React через компонент-обгортку.

Директорія `frontend/src/components/analysis/` містить компоненти для аналізу проблем з якістю, включаючи `ProblemsList.jsx` для відображення списку виявлених проблем та `ProblemDetails.jsx` для детального перегляду окремої проблеми. Компонент `ServiceComparison.jsx` забезпечує порівняльний аналіз різних сервісів IP-телефонії.

Для тестування програмного забезпечення використовуються різні типи тестів, розміщені в директоріях `backend/*/tests/` та `frontend/src/__tests__`. Модульні тести використовують Jest для JavaScript/TypeScript та pytest для Python. Інтеграційні тести перевіряють взаємодію між компонентами системи, а e2e-тести з використанням Cypress перевіряють сценарії взаємодії користувача з системою.

Конфігурація системи зберігається в файлах `.env` для кожного мікросервісу та в централізованому сховищі `etcd`. Для локальної розробки використовуються файли `.env.development`, а для продакшн-середовища - `.env.production`. Змінні оточення включають параметри підключення до баз даних, URL-адреси сервісів, параметри JWT-автентифікації та інші конфігураційні параметри.

Робочий процес розгортання програмного забезпечення автоматизований за допомогою CI/CD конвеєрів, реалізованих з використанням GitLab CI. Файл `.gitlab-ci.yml` у кореневому каталозі проєкту визначає етапи збірки, тестування та розгортання системи. Збірка Docker-образів здійснюється за допомогою `Dockerfile` в кожній директорії мікросервісу, а розгортання в Kubernetes - за допомогою Helm-чартів у директорії `kubernetes/charts/`.

3.4 Висновки за розділом 3

У третьому розділі здійснено детальний опис програмного забезпечення для оцінки сервісів IP-телефонії, реалізованого на основі спроектованої архітектури та алгоритмів.

Розроблено ефективну структуру баз даних для зберігання та обробки різних типів даних: часові ряди метрик IP-телефонії зберігаються в InfluxDB, що забезпечує високу продуктивність запису метрик; результати аналізу та виявлені проблеми документуються в MongoDB з гнучкою схемою даних; конфігурації системи зберігаються в `etcd`, що забезпечує високу доступність та узгодженість конфігураційних параметрів. Для підвищення ефективності доступу до даних створено оптимізовані індекси за основними атрибутами, що дозволяє швидко отримувати результати за типовими сценаріями використання.

Інтерфейс користувача розроблено з використанням сучасних вебтехнологій (React, Redux) та принципів User-Centered Design, що

забезпечує зручність використання системи різними категоріями користувачів. Інтерфейс включає набір інтерактивних дашбордів для візуалізації метрик якості, систему сповіщень про виявлені проблеми та інструменти для аналізу даних. Особлива увага приділена адаптивності інтерфейсу для різних пристроїв та оптимізації його продуктивності через механізми кешування даних, віртуалізації списків та відкладеного завантаження компонентів.

Проаналізовано структуру програмного забезпечення, де серверна частина реалізована як набір мікросервісів, кожен з яких відповідає за окрему функціональність системи: API-шлюз обробляє запити від клієнтів; сервіс збору даних аналізує мережевий трафік та отримує метрики; сервіс аналізу обчислює показники якості та виявляє аномалії; сервіс зберігання забезпечує взаємодію з базами даних. Клієнтська частина реалізована як односторінковий додаток на React з комплексною системою управління станом через Redux.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Експлуатація програмного забезпечення

Розроблене програмне забезпечення для оцінки сервісів ІР-телефонії надає користувачам комплексний інструментарій для моніторингу, аналізу та оптимізації якості VoIP-систем. Експлуатація програмного забезпечення починається з процесу розгортання системи, який може бути здійснений з використанням Docker-контейнерів та оркестрації Kubernetes, що забезпечує гнучкість та масштабованість рішення. Після успішного розгортання, користувачі отримують доступ до вебінтерфейсу через будь-який сучасний браузер, де здійснюється авторизація з використанням захищеного протоколу OAuth 2.0.

Головний інтерфейс програмного забезпечення, представлений на рисунку 4.1, розділений на функціональні області: навігаційне меню, що забезпечує доступ до основних модулів системи; робоча область для відображення дашбордів та звітів; та інформаційну панель, яка висвітлює критичні події та сповіщення. Дизайн інтерфейсу розроблено з урахуванням принципів інтуїтивності та адаптивності, що дозволяє ефективно використовувати систему на різних пристроях.

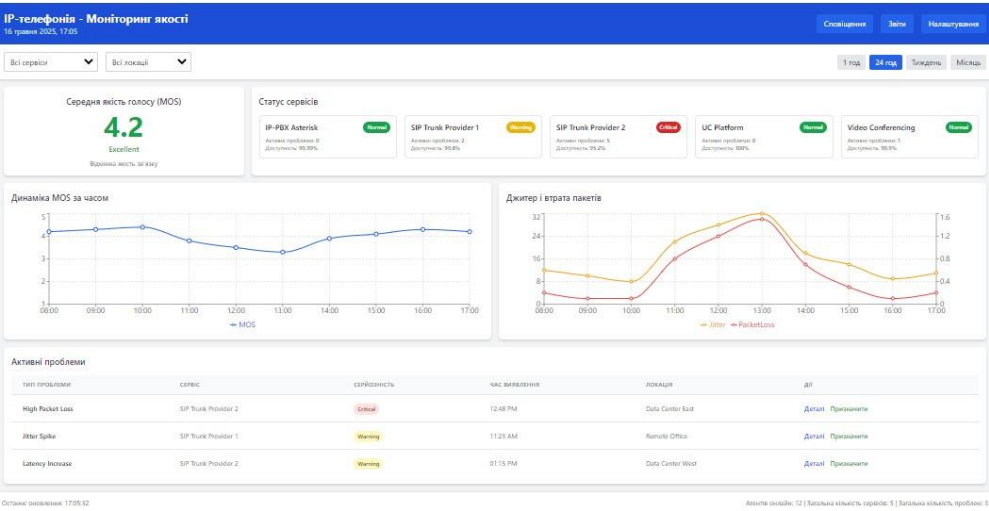


Рисунок 4.1 - Головний інтерфейс програмного забезпечення

Основним робочим інструментом користувача є модуль моніторингу, який відображає ключові метрики якості IP-телефонії в режимі реального часу. Інтерактивні графіки, реалізовані з використанням D3.js, дозволяють відслідковувати динаміку зміни параметрів мережі (затримка, джитер, втрата пакетів) та розрахованих метрик якості голосу (MOS, R-фактор). Користувач має можливість налаштувати часовий діапазон відображення даних, обрати конкретні сервіси для моніторингу та встановити порогові значення для автоматичних сповіщень. Інформаційна панель, показана на рисунку 4.2, надає швидкий доступ до агрегованих показників якості для всіх контрольованих сервісів.

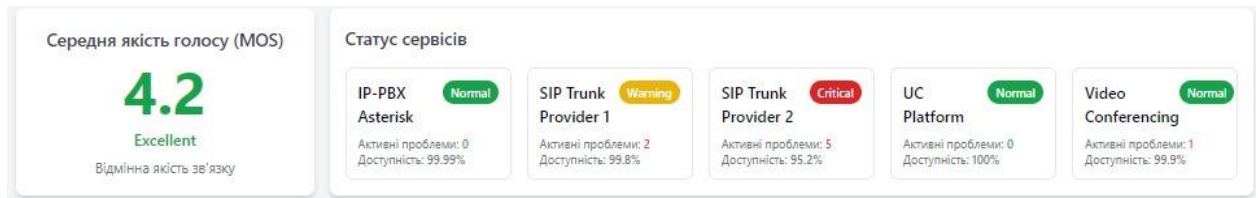


Рисунок 4.2 - Інформаційна панель з агрегованими показниками якості

Для детального аналізу якості окремого сервісу IP-телефонії, користувач може перейти до модуля сервісної аналітики, показаного на рисунках 4.3-4.4. Цей модуль надає розширену інформацію про роботу сервісу, включаючи розподіл значень метрик, статистику дзвінків, виявлені проблеми та їх можливі причини. Діагностика проблем здійснюється на основі алгоритмів машинного навчання та правил.



Рисунок 4.3 - Модуль сервісної аналітики

Активні проблеми

ТИП ПРОБЛЕМИ	СЕРВІС	СЕРІОЗНІСТЬ	ЧАС ВИЯВЛЕННЯ	ЛОКАЦІЯ	ДІЇ
High Packet Loss	SIP Trunk Provider 2	Critical	12:48 PM	Data Center East	Деталі Призначити
Jitter Spike	SIP Trunk Provider 1	Warning	11:23 AM	Remote Office	Деталі Призначити
Latency Increase	SIP Trunk Provider 2	Warning	01:15 PM	Data Center West	Деталі Призначити

Рисунок 4.4 - Модуль сервісної аналітики

Користувачі можуть налаштувати автоматичну генерацію та розсилку звітів про якість сервісів ІР-телефонії, вибравши необхідні метрики, формат звіту та розклад розсилки. Система також підтримує інтеграцію з системами управління інцидентами (ServiceNow, JIRA) для автоматичного створення завдань при виявленні проблем з якістю (рис. 4.5).



Рисунок 4.5 – Модуль звітів

4.2 Тестування основних функцій програмного забезпечення

Тестування програмного забезпечення для оцінки сервісів ІР-телефонії проводилось у кілька етапів з метою верифікації коректності роботи всіх основних функцій системи. Процес тестування охоплював

модульне тестування окремих компонентів, інтеграційне тестування взаємодії між компонентами та системне тестування всього програмного комплексу в цілому.

На етапі модульного тестування перевірялася коректність роботи ключових алгоритмів, зокрема обчислення мережових метрик (затримка, джитер, втрата пакетів) та оцінки якості голосу (MOS, R-фактор). Для автоматизації тестування використовувалися фреймворки Jest для JavaScript-компонентів та pytest для Python-модулів. Результати модульного тестування показали високу точність обчислення метрик: середня похибка для оцінки MOS не перевищувала 0,1 бала при порівнянні з еталонними значеннями.

Інтеграційне тестування було спрямоване на перевірку взаємодії між різними мікросервісами системи: від збору даних агентами до їх аналізу та візуалізації у вебінтерфейсі. Особлива увага приділялася перевірці коректності передачі даних між компонентами та стійкості системи до часткових відмов окремих сервісів. Тестування показало, що система успішно забезпечує цілісність даних та продовжує функціонувати в умовах тимчасової недоступності окремих компонентів завдяки механізмам буферизації та повторних спроб.

Функціональне тестування вебінтерфейсу проводилось як вручну, так і з використанням автоматизованих інструментів (Cypress, Selenium). Перевірялася коректність відображення даних на дашбордах, робота інтерактивних елементів, генерація звітів та експорт даних. Тестування підтвердило працездатність всіх ключових функцій інтерфейсу та відповідність дизайну вимогам до адаптивності та доступності.

Результати комплексного тестування програмного забезпечення для оцінки сервісів IP-телефонії показали високу надійність, продуктивність та функціональну повноту системи, що дозволяє рекомендувати її для практичного використання в організаціях різного масштабу для моніторингу та оптимізації якості VoIP-комунікацій.

4.3 Висновки за розділом 4

У четвертому розділі розглянуто практичні аспекти експлуатації та тестування розробленого програмного забезпечення для оцінки сервісів ІР-телефонії, що дозволило підтвердити його функціональність та відповідність встановленим вимогам.

Представлено докладний опис процесу експлуатації програмного забезпечення, включаючи розгортання системи з використанням контейнеризації та взаємодію користувача з інтерфейсом. Розглянуто основні модулі системи, які забезпечують моніторинг ключових метрик якості ІР-телефонії в режимі реального часу, детальний аналіз сервісів, порівняльний аналіз різних рішень та прогностичну аналітику. Особливу увагу приділено інтуїтивності та адаптивності інтерфейсу, що дозволяє ефективно використовувати систему на різних пристроях представниками різних категорій користувачів.

Процес тестування програмного забезпечення включав модульне тестування окремих компонентів, інтеграційне тестування взаємодії між компонентами та системне тестування всього програмного комплексу. Результати тестування підтвердили високу точність обчислення метрик, стійкість системи до часткових відмов окремих сервісів, здатність обробляти значні обсяги даних та відповідність вимогам до безпеки.

Функціональне тестування вебінтерфейсу підтвердило коректність відображення даних на дашбордах, роботу інтерактивних елементів, генерацію звітів та експорт даних. Система також успішно пройшла тестування безпеки, яке включало перевірку механізмів автентифікації та авторизації, захисту від поширених вебвразливостей та аудит коду.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було розроблено програмне забезпечення для оцінки сервісів IP-телефонії, яке дозволяє підвищити точність та оперативність оцінювання якості VoIP-систем шляхом комплексного аналізу різних аспектів їх функціонування.

У рамках дослідження було проведено аналіз технологій IP-телефонії, методів оцінки їх якості та існуючих програмних рішень. Визначено, що для комплексної оцінки необхідно враховувати множину параметрів, включаючи мережеві характеристики (затримка, джитер, втрата пакетів), якість голосу (MOS, R-фактор), надійність сервісу та функціональні можливості. Встановлено, що існуючі програмні рішення часто фокусуються на окремих аспектах оцінки без комплексного підходу, мають обмежені можливості для порівняльного аналізу різних сервісів та недостатньо використовують методи машинного навчання.

На основі проведеного аналізу було сформульовано вимоги до розроблюваного програмного забезпечення, спроектовано мікросервісну архітектуру системи, яка забезпечує модульність, масштабованість та гнучкість. Архітектура включає чотири основні рівні: інтерфейс, оркестрацію, сервіси (збору даних, аналізу, зберігання) та сховища даних. Обґрунтовано вибір технологій для реалізації різних компонентів системи, включаючи React для інтерфейсу, Kubernetes для оркестрації, InfluxDB для зберігання метрик та Python з бібліотеками машинного навчання для аналізу даних.

Розроблено алгоритми оцінки сервісів IP-телефонії, включаючи алгоритм обчислення оцінки MOS на основі E-моделі, алгоритми виявлення аномалій та прогнозування якості з використанням методів машинного навчання. Запропоновано інтегральний показник для комплексної оцінки якості, який враховує різні аспекти функціонування сервісів IP-телефонії з

можливістю налаштування вагових коефіцієнтів відповідно до пріоритетів конкретної організації.

Створено ефективну структуру баз даних для зберігання та обробки різних типів даних, включаючи часові ряди метрик в InfluxDB, результати аналізу та виявлені проблеми в MongoDB. Розроблено інтерфейс користувача з використанням принципів User-Centered Design, що забезпечує зручність використання системи.

Проведено тестування розробленого програмного забезпечення, яке підтвердило високу точність обчислення метрик, стійкість системи до часткових відмов окремих сервісів, здатність обробляти значні обсяги даних та відповідність вимогам до безпеки.

Практична значущість роботи полягає у створенні інструментарію, який дозволяє адміністраторам мереж та IT-менеджерам об'єктивно оцінювати якість різних сервісів IP-телефонії, оперативно виявляти проблеми та оптимізувати конфігурації для забезпечення стабільного функціонування комунікаційних систем підприємств. Розроблене програмне забезпечення може бути використане в організаціях різного масштабу для моніторингу та оптимізації якості VoIP-комунікацій.

Подальший розвиток роботи може бути спрямований на розширення функціональності системи для оцінки якості інших типів комунікацій (відеоконференції, обмін повідомленнями), вдосконалення алгоритмів прогнозування з використанням більш складних моделей машинного навчання, та інтеграцію з системами автоматичного управління мережевою інфраструктурою для автоматичної оптимізації параметрів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ITU-T Recommendation G.107. The E-model: A computational model for use in transmission planning. Geneva: ITU, 2015. 82 p.
2. ITU-T Recommendation P.862. Perceptual Evaluation of Speech Quality (PESQ). Geneva: ITU, 2001. 68 p.
3. ITU-T Recommendation P.863. Perceptual Objective Listening Quality Analysis (POLQA). Geneva: ITU, 2014. 95 p.
4. Biernacki C., Piskorski J., Duflot M. Anomaly detection in VoIP traffic: Overview and evaluation of methods. Computer Communications, Vol. 39, 2014, pp. 1–23.
5. Barford P., Sommers J. Comparative Evaluation of Passive and Active Traffic Measurement Systems. In: Proc. of ACM SIGCOMM IMC, 2004. p. 257–270.
6. Dressler F., Karnouskos S. From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Elsevier, 2014. 140 p.
7. Varga A., Hornig R. An overview of the OMNeT++ simulation environment. In: Proc. of the 1st Int. Conf. on Simulation Tools and Techniques, 2008. 10 p.
8. Sun W., Li F., Srivastava J. Detecting VoIP flooding attacks using entropy-based measures. In: IEEE Globecom Workshops, 2007. pp. 1–6.
9. Pathak P.H., Hu Y.C., Zhang M. Measurement and Analysis of Skype Video Calls: Performance, Usability, and the Network Effect. In: IEEE INFOCOM, 2012. pp. 1112–1120.
10. Quality of Experience (QoE) Working Group. QoE Assessment Methodologies for VoIP Applications. IETF Draft, 2016.
11. Cisco Systems. Voice over IP - Per Call Bandwidth Consumption. Cisco White Paper, 2018. 14 p.
12. Makhijani K., Huang L. VoIP Security and Quality Performance Metrics. In: IEEE Communications Magazine, Vol. 45, No. 1, 2007. pp. 94–101.

13. Fortino G., Trunfio P. (Eds.). Internet of Things Based on Smart Objects: Technology, Middleware and Applications. Springer, 2014. 198 p.
14. Vuong S., Chan A., Wang J. Adaptive QoS routing for real-time traffic in mobile ad hoc networks. In: Computer Communications, Vol. 27, 2004, pp. 1644–1653.
15. ITU-T Recommendation Y.1540. IP packet transfer and availability performance parameters. Geneva: ITU, 2011.
16. Tanenbaum A.S., Wetherall D.J. Computer Networks. 5th ed. Boston: Pearson, 2010. 960 p.
17. White R., Donnelly J. VoIP Performance Management and Optimization. Cisco Press, 2009. 352 p.
18. Kurose J.F., Ross K.W. Computer Networking: A Top-Down Approach. 7th ed. Boston: Pearson, 2016. 856 p.
19. Gunther N.J. Analyzing Computer System Performance with Perl::PDQ. Springer, 2011. 296 p.
20. Kim H., Feamster N. Improving network management with software defined networking. IEEE Communications Magazine, Vol. 51, No. 2, 2013. pp. 114–119.
21. Sipponen M., Suomalainen T. Evaluating VoIP Quality with Machine Learning Techniques. In: Journal of Communications, Vol. 12, No. 3, 2017. pp. 174–181.
22. Mitola J., Maguire G.Q. Cognitive radio: making software radios more personal. IEEE Personal Communications, Vol. 6, No. 4, 1999. pp. 13–18.
23. Bejou D. Artificial Intelligence in Higher Education: Applications, Promise and Perils, and Ethical Questions. Journal of Relationship Marketing, Vol. 17, No. 1, 2018. pp. 1–12.
24. ITU-T Recommendation G.113. Transmission impairments due to speech processing. Geneva: ITU, 2016.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення призначене для комплексної оцінки сервісів IP-телефонії, моніторингу параметрів якості, діагностики проблем та оптимізації VoIP-систем в організаціях різного масштабу. Воно забезпечує збір та аналіз мережевих метрик, оцінку якості голосу, виявлення аномалій та візуалізацію результатів через інтуїтивний вебінтерфейс.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для оцінки сервісів IP-телефонії», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Призначення розробки

Програмний продукт призначений для автоматизації процесів моніторингу та аналізу якості сервісів IP-телефонії, виявлення проблем та аномалій, порівняльного аналізу різних VoIP-рішень та підтримки прийняття рішень щодо оптимізації та вибору сервісів IP-телефонії.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для оцінки сервісів IP-телефонії забезпечує виконання таких функцій:

- захоплення та аналіз мережевого трафіку IP-телефонії (SIP, RTP, H.323);
- обчислення мережевих метрик (затримка, джитер, втрата пакетів);

- обчислення метрик якості голосу (MOS, R-фактор);
- виявлення аномалій та проблем у роботі сервісів IP-телефонії;
- генерація звітів;;
- сповіщення про виявлені проблеми через різні канали комунікації.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення повинен бути зручним для користувачів різних категорій (адміністратори мереж, ІТ-менеджери, керівники). Він має бути інтуїтивно зрозумілим, адаптивним для різних пристроїв (десктоп, планшет, смартфон) та відповідати сучасним стандартам вебдизайну. Інтерфейс повинен забезпечувати інтерактивну візуалізацію метрик у вигляді графіків, діаграм та дашбордів з можливістю налаштування відображення даних.

A.3.3 Вимоги до надійності

Програмне забезпечення повинно забезпечувати надійне функціонування в режимі 24/7 з доступністю не менше 99.9% часу. Система має забезпечувати стійкість до часткових відмов окремих компонентів, збереження даних при збоях, автоматичне відновлення після збоїв та можливість резервного копіювання та відновлення даних.

A.3.4 Умови експлуатації

Для експлуатації програмного забезпечення необхідна наявність серверної інфраструктури з можливістю розгортання контейнеризованих додатків та доступом до мережевого трафіку IP-телефонії. Клієнтська частина може використовуватись через будь-який сучасний веббраузер з підтримкою HTML5, CSS3 та JavaScript.

A.3.5 Вимоги до складу та параметрів технічних засобів

Для роботи програмного забезпечення потрібні такі технічні та програмні ресурси:

Серверна частина:

- сервер з процесором не менше 8 ядер;
- оперативна пам'ять не менше 16 ГБ;
- дисковий простір не менше 500 ГБ;
- kubernetes кластер або Docker для контейнеризації;
- influxDB для зберігання метрик;
- mongoDB для зберігання результатів аналізу;
- Apache Kafka для обміну повідомленнями;
- etcd для зберігання конфігурацій.

Клієнтська частина:

- сучасний браузер (Chrome, Firefox, Safari, Edge останніх версій);
- JavaScript включений;
- доступ до мережі, де розгорнуто серверну частину.

A.3.6 Вимоги до маркування і пакування

Програмне забезпечення для оцінки сервісів IP-телефонії може бути розгорнуто в Kubernetes кластері або з використанням Docker Compose через систему контролю версій Git.

A.4 Вхідні та вихідні дані

Вхідними даними до програмного забезпечення є:

- мережевий трафік IP-телефонії (SIP, RTP, H.323);
- дані з API різних систем IP-телефонії;
- статистика з мережевого обладнання;

- результати синтетичних тестів;
- конфігураційні параметри для налаштування алгоритмів аналізу.

Вихідними даними програмного забезпечення є:

- візуалізація метрик якості в інтерфейсі;
- звіти про якість сервісів ІР-телефонії;
- сповіщення про виявлені проблеми та аномалії.

ДОДАТОК Б
Фрагмент тексту програми

Лістинг Б.1 – Реалізація алгоритму обчислення оцінки MOS на основі

Е-моделі

```

* Обчислення оцінки MOS (Mean Opinion Score) на основі Е-моделі (ITU-T
G.107)
* @param {number} latency - затримка в мілісекундах
* @param {number} jitter - джитер в мілісекундах
* @param {number} packetLoss - втрата пакетів у відсотках
* @param {string} codec - тип кодека (G.711, G.729, ...)
* @returns {number} оцінка MOS за шкалою від 1 до 5
*/
function calculateMOS(latency, jitter, packetLoss, codec) {
  // Визначення базових параметрів Е-моделі для різних кодеків
  const codecParams = {
    'G.711': { ie: 0, bpl: 4.3 },
    'G.729': { ie: 10, bpl: 19 },
    'G.723.1-6.3k': { ie: 15, bpl: 15 },
    'G.723.1-5.3k': { ie: 19, bpl: 14 },
    'G.726-32k': { ie: 7, bpl: 23 },
    'G.726-24k': { ie: 16, bpl: 23 },
    'G.728': { ie: 7, bpl: 20 },
    'GSM-FR': { ie: 20, bpl: 10 },
    'GSM-HR': { ie: 23, bpl: 9 },
    'GSM-EFR': { ie: 5, bpl: 10 },
    'OPUS': { ie: 4, bpl: 10 }
  };

  // Значення за замовчуванням, якщо кодек не знайдено
  const { ie, bpl } = codecParams[codec] || { ie: 10, bpl: 10 };

  // Базовий R0 фактор
  const R0 = 93.2;

  // Розрахунок погіршення через затримку (Id)
  let Id = 0;
  if (latency < 100) {
    Id = 0;
  } else if (latency < 400) {
    Id = 0.024 * latency + 0.11 * (latency - 177.3) * (latency > 177.3
? 1 : 0);
  } else {
    Id = 44;
  }

  // Врахування впливу джитера через збільшення ефективної втрати
пакетів
  const effectivePacketLoss = packetLoss + (jitter > 30 ? 2 : jitter /
15);

  // Розрахунок погіршення через втрату пакетів та кодек (Ie_eff)
  const Ie_eff = ie + (95 - ie) * (effectivePacketLoss /
(effectivePacketLoss + bpl));

  // Фактор очікування (A)
  const A = 0; // Для більшості випадків дорівнює 0

```

```

// Обчислення R-фактора
const R = R0 - Id - Ie_eff + A;

// Конвертація R-фактора в MOS
let MOS;
if (R < 0) {
  MOS = 1.0;
} else if (R > 100) {
  MOS = 4.5;
} else {
  MOS = 1 + 0.035 * R + R * (R - 60) * (100 - R) * 7e-6;
}

// Округлення до десятих
return parseFloat(MOS.toFixed(1));
}

const latency = 120; // мс
const jitter = 15; // мс
const packetLoss = 1.5; // %
const codec = 'G.711';

const mosScore = calculateMOS(latency, jitter, packetLoss, codec);
console.log(`MOS оцінка: ${mosScore}`);

```

Лістинг Б.2 – Модуль збору мережевих метрик IP-телефонії

```

import * as pcap from 'pcap';
import * as dgram from 'dgram';
import { Kafka } from 'kafkajs';

interface NetworkMetrics {
  timestamp: number;
  serviceId: string;
  sourceIp: string;
  destinationIp: string;
  codec: string;
  latency: number;
  jitter: number;
  packetLoss: number;
  bandwidth: number;
}

class VoipMetricsCollector {
  private pcapSession: pcap.Session;
  private packets: Map<string, any[]> = new Map();
  private sequenceNumbers: Map<string, number[]> = new Map();
  private lastPacketTime: Map<string, number> = new Map();
  private kafka: Kafka;
  private producer: any;

  constructor(private interface: string, private serviceId: string,
    private kafkaBrokers: string[]) {
    // Ініціалізація захоплення пакетів

```

```

    this.pcapSession = pcap.createSession(interface, 'udp port 5060 or
udp portrange 10000-20000');

    // Ініціалізація Kafka
    this.kafka = new Kafka({
      clientId: 'voip-metrics-collector',
      brokers: this.kafkaBrokers
    });

    this.producer = this.kafka.producer();
  }

  public async start() {
    await this.producer.connect();

    console.log(`Розпочато збір метрик для сервісу ${this.serviceId}
на інтерфейсі ${this.interface}`);

    this.pcapSession.on('packet', (rawPacket) => {
      const packet = pcap.decode.packet(rawPacket);

      // Перевірка, чи це RTP пакет
      if (this.isRtpPacket(packet)) {
        this.processRtpPacket(packet);
      }

      // Перевірка, чи це SIP пакет
      if (this.isSipPacket(packet)) {
        this.processSipPacket(packet);
      }
    });

    // Періодичне обчислення метрик
    setInterval(() => this.calculateAndSendMetrics(), 5000);
  }

  private isRtpPacket(packet: any): boolean {
    try {
      const udp = packet.payload.payload;
      // RTP версія повинна бути 2 (перші два біти)
      return (udp.data[0] & 0xC0) === 0x80;
    } catch (error) {
      return false;
    }
  }

  private isSipPacket(packet: any): boolean {
    try {
      const udp = packet.payload.payload;
      const data = udp.data.toString('utf8');
      return data.startsWith('SIP/2.0') ||
        data.startsWith('INVITE') ||
        data.startsWith('REGISTER');
    } catch (error) {
      return false;
    }
  }
}

```

```

private processRtpPacket(packet: any) {
  try {
    const ip = packet.payload;
    const udp = ip.payload;
    const data = udp.data;

    const sourceIp = ip.saddr.join('.');
    const destIp = ip.daddr.join('.');
    const streamId = `${sourceIp}:${udp.sport}-${destIp}:${udp.dport}`;

    // Витягуємо RTP дані
    const sequenceNumber = (data[2] << 8) | data[3];
    const timestamp = packet.pcap_header.tv_sec * 1000 +
packet.pcap_header.tv_usec / 1000;
    const payloadType = data[1] & 0x7F;

    // Зберігаємо пакет для аналізу
    if (!this.packets.has(streamId)) {
      this.packets.set(streamId, []);
      this.sequenceNumbers.set(streamId, []);
      this.lastPacketTime.set(streamId, timestamp);
    }

    this.packets.get(streamId)!.push({
      sequenceNumber,
      timestamp,
      payloadType,
      size: data.length
    });

    this.sequenceNumbers.get(streamId)!.push(sequenceNumber);

    // Обмежуємо розмір буферів для уникнення витоків пам'яті
    if (this.packets.get(streamId)!.length > 1000) {
      this.packets.get(streamId)!.shift();
      this.sequenceNumbers.get(streamId)!.shift();
    }
  } catch (error) {
    console.error('Помилка обробки RTP пакета:', error);
  }
}

private processSipPacket(packet: any) {
  // Обробка SIP-пакетів для виявлення кодеків і встановлення сесій
  // Реалізація залежить від потреб моніторингу
}

private calculateAndSendMetrics() {
  for (const [streamId, packets] of this.packets.entries()) {
    if (packets.length < 2) continue;

    const [sourceIp, destIp] = streamId.split('-').map(ip =>
ip.split(':')[0]);

    // Обчислення затримки (приблизно, на основі RTT)
    const delays = [];
    for (let i = 1; i < packets.length; i++) {

```

```

        delays.push(packets[i].timestamp - packets[i-1].timestamp);
    }
    const latency = delays.reduce((sum, delay) => sum + delay, 0) /
delays.length / 2;

    // Обчислення джитера
    const jitterValues = [];
    for (let i = 1; i < delays.length; i++) {
        jitterValues.push(Math.abs(delays[i] - delays[i-1]));
    }
    const jitter = jitterValues.length > 0 ?
        jitterValues.reduce((sum, jitter) => sum + jitter, 0) /
jitterValues.length : 0;

    // Обчислення втрати пакетів
    const seqNums = this.sequenceNumbers.get(streamId)!;
    const expectedPackets = seqNums[seqNums.length - 1] - seqNums[0]
+ 1;
    const receivedPackets = seqNums.length;
    const packetLoss = expectedPackets > 0 ?
        ((expectedPackets - receivedPackets) / expectedPackets) * 100
: 0;

    // Обчислення пропускної здатності
    const totalBytes = packets.reduce((sum, packet) => sum +
packet.size, 0);
    const timeSpan = (packets[packets.length - 1].timestamp -
packets[0].timestamp) / 1000; // в секундах
    const bandwidth = timeSpan > 0 ? (totalBytes * 8 / timeSpan) /
1000 : 0; // в кбіт/с

    // Визначення кодека (спрощено)
    let codec = 'Unknown';
    const payloadType = packets[0].payloadType;
    if (payloadType === 0) codec = 'G.711-PCMU';
    else if (payloadType === 8) codec = 'G.711-PCMA';
    else if (payloadType === 18) codec = 'G.729';
    else if (payloadType === 4) codec = 'G.723.1';
    else if (payloadType === 111) codec = 'OPUS';

    // Формування метрик
    const metrics: NetworkMetrics = {
        timestamp: Date.now(),
        serviceId: this.serviceId,
        sourceIp,
        destinationIp: destIp,
        codec,
        latency,
        jitter,
        packetLoss,
        bandwidth
    };

    // Відправка метрик у Kafka
    this.sendToKafka(metrics);
}
}

```

```

private async sendToKafka(metrics: NetworkMetrics) {
  try {
    await this.producer.send({
      topic: 'voip-network-metrics',
      messages: [
        {
          key: `${metrics.serviceId}-${metrics.sourceIp}-${metrics.destinationIp}`,
          value: JSON.stringify(metrics)
        }
      ],
    });
  } catch (error) {
    console.error('Помилка відправки в Kafka:', error);
  }
}

public stop() {
  this.pcapSession.close();
  this.producer.disconnect();
  console.log(`Збір метрик зупинено для сервісу ${this.serviceId}`);
}
}

const collector = new VoipMetricsCollector('eth0', 'voip-service-1',
['kafka:9092']);
collector.start().catch(console.error);

// Обробка завершення роботи
process.on('SIGINT', () => {
  collector.stop();
  process.exit(0);
});

```

Лістинг Б.3 – Реалізація компонента візуалізації метрик якості IP-телефонії

```

import React, { useState, useEffect } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip,
Legend, ResponsiveContainer } from 'recharts';
import { fetchMetrics } from '../redux/actions/metricsActions';
import { Card, CardHeader, CardContent, Grid, Typography, Box,
FormControl, InputLabel, Select, MenuItem } from '@mui/material';

const MetricsChart = ({ serviceId, startTime, endTime }) => {
  const dispatch = useDispatch();
  const { metrics, loading, error } = useSelector(state =>
state.metrics);
  const [selectedMetric, setSelectedMetric] = useState('mos');

  // Доступні метрики для відображення
  const availableMetrics = [

```

```

    { id: 'mos', name: 'MOS', color: '#2196f3', domain: [1, 5],
      threshold: 3.5 },
    { id: 'latency', name: 'Затримка (мс)', color: '#f44336', domain:
      [0, 400], threshold: 150 },
    { id: 'jitter', name: 'Джитер (мс)', color: '#ff9800', domain: [0,
      100], threshold: 30 },
    { id: 'packetLoss', name: 'Втрата пакетів (%)', color: '#4caf50',
      domain: [0, 10], threshold: 1 }
  ];

  // Функція для визначення статусу метрики
  const getMetricStatus = (value, metricId) => {
    const metric = availableMetrics.find(m => m.id === metricId);
    if (!metric) return 'normal';

    if (metricId === 'mos') {
      return value < metric.threshold ? 'critical' : 'normal';
    } else {
      return value > metric.threshold ? 'critical' : 'normal';
    }
  };

  // Перетворення даних метрик для відображення на графіку
  const formatMetricsForChart = () => {
    if (!metrics || !metrics.length) return [];

    return metrics.map(metric => ({
      timestamp: new Date(metric.timestamp).toLocaleTimeString(),
      mos: metric.mos,
      latency: metric.latency,
      jitter: metric.jitter,
      packetLoss: metric.packetLoss,
      mosStatus: getMetricStatus(metric.mos, 'mos'),
      latencyStatus: getMetricStatus(metric.latency, 'latency'),
      jitterStatus: getMetricStatus(metric.jitter, 'jitter'),
      packetLossStatus: getMetricStatus(metric.packetLoss,
'packetLoss')
    })));
  };

  useEffect(() => {
    dispatch(fetchMetrics(serviceId, startTime, endTime));

    // Оновлення даних кожні 30 секунд
    const interval = setInterval(() => {
      dispatch(fetchMetrics(serviceId, startTime, endTime));
    }, 30000);

    return () => clearInterval(interval);
  }, [dispatch, serviceId, startTime, endTime]);

  // Отримання поточного значення метрики
  const getCurrentValue = () => {
    if (!metrics || !metrics.length) return 'N/A';
    const lastMetric = metrics[metrics.length - 1];
    return lastMetric[selectedMetric]?.toFixed(2) || 'N/A';
  };

```

```

// Отримання інформації про обрану метрику
const getSelectedMetricInfo = () => {
  return availableMetrics.find(m => m.id === selectedMetric) ||
availableMetrics[0];
};

// Обробка зміни обраної метрики
const handleMetricChange = (event) => {
  setSelectedMetric(event.target.value);
};

const currentValue = getCurrentValue();
const selectedMetricInfo = getSelectedMetricInfo();
const metricStatus = getMetricStatus(currentValue, selectedMetric);

if (loading) return <Typography>Завантаження метрик...</Typography>;
if (error) return <Typography color="error">Помилка завантаження
метрик: {error}</Typography>;

const chartData = formatMetricsForChart();

return (
  <Card>
    <CardHeader
      title="Метрики якості IP-телефонії"
      subheader={`Сервіс: ${serviceId}`}
      action={
150   <FormControl variant="outlined" size="small" sx={{ minWidth:
        <InputLabel>Метрика</InputLabel>
        <Select
          value={selectedMetric}
          onChange={handleMetricChange}
          label="Метрика"
        >
          {availableMetrics.map(metric => (
            <MenuItem key={metric.id}
value={metric.id}>{metric.name}</MenuItem>
          ))}
        </Select>
      </FormControl>
    }
  </>
  <CardContent>
    <Grid container spacing={3}>
      <Grid item xs={12} md={3}>
        <Box
          sx={{
            p: 2,
            backgroundColor: metricStatus === 'critical' ?
'#ffebee' : '#e8f5e9',
            borderRadius: 1,
            textAlign: 'center'
          }}
        >
          <Typography
variant="caption">{selectedMetricInfo.name}</Typography>
          <Typography

```

```

        variant="h4"
        color={metricStatus === 'critical' ? 'error' :
'success'}
    >
        {currentValue}
    </Typography>
    <Typography variant="caption">
        Попир: {selectedMetricInfo.threshold}
        {selectedMetric === 'mos' ? ' (мін)' : ' (макс)'}
    </Typography>
    </Box>
</Grid>
<Grid item xs={12} md={9}>
    <Box sx={{ height: 300 }}>
        <ResponsiveContainer width="100%" height="100%">
            <LineChart
                data={chartData}
                margin={{ top: 5, right: 30, left: 20, bottom: 5 }}
            >
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis
                    dataKey="timestamp"
                    label={{ value: 'Час', position:
'insideBottomRight', offset: -10 }}
                />
                <YAxis
                    domain={selectedMetricInfo.domain}
                    label={{
                        value: selectedMetricInfo.name,
                        angle: -90,
                        position: 'insideLeft'
                    }}
                />
                <Tooltip />
                <Legend />
                <Line
                    type="monotone"
                    dataKey={selectedMetric}
                    stroke={selectedMetricInfo.color}
                    activeDot={{ r: 8 }}
                    strokeWidth={2}
                />
                {/* Лінія порогового значення */}
                <Line
                    type="monotone"
                    dataKey={`_${selectedMetric}Threshold`}
                    stroke="#9e9e9e"
                    strokeDasharray="5 5"
                    strokeWidth={1}
                    dot={false}
                    activeDot={false}
                    isAnimationActive={false}
                />
            </LineChart>
        </ResponsiveContainer>
    </Box>
</Grid>
</Grid>

```

```

        </CardContent>
    </Card>
);
};

export default MetricsChart;

```

Лістинг Б.4 – Сервіс аналізу та виявлення аномалій в метриках IP-телефонії

```

import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from confluent_kafka import Consumer, Producer
import json
import logging
from datetime import datetime, timedelta
import pymongo
from influxdb import InfluxDBClient

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

class VoipAnomalyDetector:
    def __init__(self, kafka_config, influxdb_config, mongodb_config):
        # Налаштування Kafka
        self.consumer_config = {
            'bootstrap.servers': kafka_config['bootstrap_servers'],
            'group.id': 'voip-anomaly-detector',
            'auto.offset.reset': 'latest'
        }
        self.producer_config = {
            'bootstrap.servers': kafka_config['bootstrap_servers']
        }
        self.consumer = Consumer(self.consumer_config)
        self.producer = Producer(self.producer_config)
        self.consumer.subscribe([kafka_config['input_topic']])
        self.output_topic = kafka_config['output_topic']

        # Підключення до InfluxDB
        self.influxdb = InfluxDBClient(
            host=influxdb_config['host'],
            port=influxdb_config['port'],
            username=influxdb_config['username'],
            password=influxdb_config['password'],
            database=influxdb_config['database']
        )

        # Підключення до MongoDB
        self.mongodb =
pymongo.MongoClient(mongodb_config['connection_string'])

```

```

        self.db = self.mongoddb[mongoddb_config['database']]
        self.problems_collection
self.db[mongoddb_config['problems_collection']]

        # Моделі для виявлення аномалій
        self.models = {}
        self.scalers = {}

        # Параметри для аналізу
        self.window_size = 60 # Розмір вікна для аналізу (кількість
точок)
        self.contamination = 0.05 # Приблизний відсоток аномалій в
даних
        self.metrics = ['latency', 'jitter', 'packet_loss', 'mos']

        # Порогові значення для різних метрик
        self.thresholds = {
            'latency': 150, # мс
            'jitter': 30, # мс
            'packet_loss': 1.0, # %
            'mos': 3.5 # мінімальне прийнятне значення
        }

        # Кешовані дані про сервіси
        self.services_cache = {}

        logger.info("VoIP Anomaly Detector ініціалізовано")

    def load_historical_data(self, service_id, hours=24):
        """Завантаження історичних даних з InfluxDB для тренування
моделей"""
        time_from = datetime.utcnow() - timedelta(hours=hours)
        time_from_str = time_from.strftime('%Y-%m-%dT%H:%M:%SZ')

        query = f"""
            SELECT mean(latency) as latency,
                   mean(jitter) as jitter,
                   mean(packet_loss) as packet_loss,
                   mean(mos_estimate) as mos
            FROM voip_network_metrics
            WHERE service_id = '{service_id}'
            AND time > '{time_from_str}'
            GROUP BY time(5m)
            FILL(linear)
        """

        result = self.influxdb.query(query)
        if not result:
            logger.warning(f"Немає історичних даних для сервісу
{service_id}")
            return None

        df = pd.DataFrame(list(result.get_points()))
        if df.empty:
            logger.warning(f"Порожні історичні дані для сервісу
{service_id}")
            return None

```

```

# Перетворення часових міток
df['time'] = pd.to_datetime(df['time'])
df.set_index('time', inplace=True)

# Заповнення пропущених значень
df.fillna(method='ffill', inplace=True)
df.fillna(method='bfill', inplace=True)

return df

def train_models(self, service_id):
    """Тренування моделей для виявлення аномалій для конкретного
сервісу"""
    df = self.load_historical_data(service_id)
    if df is None or len(df) < self.window_size:
        logger.warning(f"Недостатньо даних для тренування моделей
для сервісу {service_id}")
        return False

    self.models[service_id] = {}
    self.scalers[service_id] = {}

    for metric in self.metrics:
        if metric not in df.columns:
            logger.warning(f"Метрика {metric} відсутня в даних для
сервісу {service_id}")
            continue

        # Підготовка даних
        X = df[[metric]].values

        # Масштабування даних
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)
        self.scalers[service_id][metric] = scaler

        # Тренування моделі
        model = IsolationForest(
            contamination=self.contamination,
            random_state=42,
            n_estimators=100
        )
        model.fit(X_scaled)

        self.models[service_id][metric] = model

    logger.info(f"Моделі для сервісу {service_id} успішно
натреновані")
    return True

def detect_anomalies(self, service_id, metrics_data):
    """Виявлення аномалій в метриках для конкретного сервісу"""
    if service_id not in self.models:
        if not self.train_models(service_id):
            # Якщо не вдалося тренувати моделі, використовуємо
порогові значення
            return
    self.detect_anomalies_by_thresholds(metrics_data)

```

```

anomalies = {}

for metric in self.metrics:
    if metric not in metrics_data or service_id not in
self.models or metric not in self.models[service_id]:
        continue

    # Визначення, чи є значення аномальним за моделлю
    value = metrics_data[metric]

    # Масштабування значення
    scaled_value =
self.scalers[service_id][metric].transform([[value]])

    # Прогнозування аномалії
    prediction =
self.models[service_id][metric].predict(scaled_value)

    # -1 означає аномалію
    is_anomaly = prediction[0] == -1

    # Також перевіряємо порогові значення
    is_threshold_violation =
self.is_threshold_violation(metric, value)

    if is_anomaly or is_threshold_violation:
        anomalies[metric] = {
            'value': value,
            'threshold': self.thresholds.get(metric),
            'is_model_anomaly': is_anomaly,
            'is_threshold_violation': is_threshold_violation
        }

return anomalies

def is_threshold_violation(self, metric, value):
    """Перевірка порушення порогових значень для метрики"""
    if metric not in self.thresholds:
        return False

    threshold = self.thresholds[metric]

    if metric == 'mos':
        return value < threshold
    else:
        return value > threshold

def detect_anomalies_by_thresholds(self, metrics_data):
    """Виявлення аномалій на основі порогових значень"""
    anomalies = {}

    for metric in self.metrics:
        if metric not in metrics_data:
            continue

        value = metrics_data[metric]

```

```

        is_threshold_violation
self.is_threshold_violation(metric, value)

        if is_threshold_violation:
            anomalies[metric] = {
                'value': value,
                'threshold': self.thresholds.get(metric),
                'is_model_anomaly': False,
                'is_threshold_violation': True
            }

        return anomalies

def analyze_anomalies(self, service_id, anomalies, metrics_data):
    """Аналіз виявлених аномалій та діагностика проблем"""
    if not anomalies:
        return None

    # Отримання інформації про сервіс
    service_info = self.get_service_info(service_id)

    # Визначення ймовірних причин проблем
    problems = []

    # Проблеми з затримкою
    if 'latency' in anomalies and 'jitter' not in anomalies and
'packet_loss' not in anomalies:
        problems.append({
            'type': 'network_latency',
            'severity': 3,
            'description': 'Висока затримка без збільшення джитера та втрати
пакетів',
            'possible_cause': 'Проблеми з маршрутизацією або велика
відстань до сервера',
            'recommendation': 'Перевірити маршрути проходження
трафіку, розглянути можливість використання серверів, географічно
ближчих до користувачів'
        })

    # Проблеми з джитером
    if 'jitter' in anomalies and 'packet_loss' not in anomalies:
        problems.append({
            'type': 'network_jitter',
            'severity': 3,
            'description': 'Високий джитер без значної втрати
пакетів',
            'possible_cause': 'Нестабільність мережі або недостатні
буфери джитера',
            'recommendation': 'Збільшити розмір буферів джитера,
налаштувати QoS для пріоритизації голосового трафіку'
        })

    # Проблеми з втратою пакетів
    if 'packet_loss' in anomalies:
        severity = 4 if metrics_data.get('packet_loss', 0) > 5 else
3
        problems.append({
            'type': 'network_packet_loss',

```

```

        'severity': severity,
        'description': f'Висока втрата пакетів:
{metrics_data.get("packet_loss", 0):.2f}%',
        'possible_cause': 'Перевантаження мережі, проблеми з
обладнанням або недостатня пропускна здатність',
        'recommendation': 'Перевірити завантаженість мережевих
інтерфейсів, збільшити пропускну здатність, налаштувати QoS'
    })

    # Комбіновані проблеми
    if 'latency' in anomalies and 'jitter' in anomalies and
'packet_loss' in anomalies:
        problems.append({
            'type': 'network_critical',
            'severity': 5,
            'description': 'Критичні проблеми з мережею: висока
затримка, джитер та втрата пакетів',
            'possible_cause': 'Серйозні проблеми з мережевою
інфраструктурою або перевантаження',
            'recommendation': 'Негайно провести комплексну
діагностику мережі та обладнання'
        })

    # Низька якість голосу
    if 'mos' in anomalies:
        mos_value = metrics_data.get('mos', 0)
        severity = 5 if mos_value < 2.5 else (4 if mos_value < 3.0
else 3)
        problems.append({
            'type': 'voice_quality',
            'severity': severity,
            'description': f'Низька якість голосу: MOS =
{mos_value:.2f}',
            'possible_cause': 'Проблеми з кодеком, мережеві
проблеми або неоптимальні налаштування',
            'recommendation': 'Перевірити налаштування кодеків,
проаналізувати мережеві параметри'
        })

    # Вибір найбільш критичної проблеми
    most_critical_problem = max(problems, key=lambda p:
p['severity']) if problems else None

    if most_critical_problem:
        # Створення запису про проблему
        problem_record = {
            'service_id': service_id,
            'service_name': service_info.get('name', service_id),
            'problem_type': most_critical_problem['type'],
            'severity': most_critical_problem['severity'],
            'description': most_critical_problem['description'],
            'possible_cause':
most_critical_problem['possible_cause'],
            'recommendation':
most_critical_problem['recommendation'],
            'detected_at': datetime.utcnow(),
            'status': 'new',
            'metrics': metrics_data,

```

```

        'anomalies': anomalies
    }

    # Перевірка, чи вже існує активна проблема цього типу
    existing_problem = self.problems_collection.find_one({
        'service_id': service_id,
        'problem_type': most_critical_problem['type'],
        'status': {'$in': ['new', 'acknowledged']}
    })

    if not existing_problem:
        self.problems_collection.insert_one(problem_record)
        logger.info(f"Виявлено нову проблему для сервісу
{service_id}: {most_critical_problem['type']}")
        return problem_record
    else:
        logger.info(f"Проблема {most_critical_problem['type']}
для сервісу {service_id} вже існує")
        return existing_problem

    return None

def get_service_info(self, service_id):
    """Отримання інформації про сервіс"""
    if service_id in self.services_cache:
        return self.services_cache[service_id]

    # Отримання інформації з MongoDB
    service = self.db.services.find_one({'service_id': service_id})

    if not service:
        # Базова інформація, якщо сервіс не знайдено
        service = {
            'service_id': service_id,
            'name': f'Сервіс {service_id}',
            'type': 'unknown'
        }

    self.services_cache[service_id] = service
    return service

def process_message(self, message):
    """Обробка повідомлення з Kafka"""
    try:
        message_value = message.value().decode('utf-8')
        data = json.loads(message_value)

        service_id = data.get('serviceId')
        if not service_id:
            logger.warning("Повідомлення не містить ідентифікатора
сервісу")
            return

        # Підготовка даних для аналізу
        metrics_data = {
            'latency': data.get('latency', 0),
            'jitter': data.get('jitter', 0),
            'packet_loss': data.get('packetLoss', 0),

```

```

        'mos': data.get('mosEstimate', 0)
    }

    # Виявлення аномалій
    anomalies = self.detect_anomalies(service_id, metrics_data)

    if anomalies:
        logger.info(f"Виявлено аномалії для сервісу
{service_id}: {anomalies}")

        # Аналіз аномалій та діагностика проблем
        problem = self.analyze_anomalies(service_id, anomalies,
metrics_data)

        if problem:
            # Відправка сповіщення
            self.send_alert(problem)

    except Exception as e:
        logger.error(f"Помилка обробки повідомлення: {e}")

def send_alert(self, problem):
    """Відправка сповіщення про виявлену проблему"""
    try:
        # Підготовка даних для сповіщення
        alert = {
            'type': 'voip_problem_detected',
            'service_id': problem['service_id'],
            'service_name': problem['service_name'],
            'problem_type': problem['problem_type'],
            'severity': problem['severity'],
            'description': problem['description'],
            'detected_at': problem['detected_at'].isoformat(),
            'problem_id': str(problem.get('_id', ''))
        }

        # Відправка сповіщення в Kafka
        self.producer.produce(
            self.output_topic,
            key=problem['service_id'],
            value=json.dumps(alert)
        )
        self.producer.flush()

        logger.info(f"Сповіщення про проблему відправлено:
{problem['problem_type']}")

    except Exception as e:
        logger.error(f"Помилка відправки сповіщення: {e}")

def run(self):
    """Основний цикл обробки повідомлень"""
    try:
        logger.info("Запуск обробки повідомлень")

        while True:
            msg = self.consumer.poll(timeout=1.0)

```

```

        if msg is None:
            continue

        if msg.error():
            logger.error(f"Помилка отримання повідомлення:
{msg.error()}")
            continue

        self.process_message(msg)

    except KeyboardInterrupt:
        logger.info("Отримано сигнал переривання, завершення
роботи")
    finally:
        self.consumer.close()
        logger.info("Споживач Kafka закритий")

if __name__ == "__main__":
    # Конфігурація
    kafka_config = {
        'bootstrap_servers': 'kafka:9092',
        'input_topic': 'voip-network-metrics',
        'output_topic': 'voip-alerts'
    }

    influxdb_config = {
        'host': 'influxdb',
        'port': 8086,
        'username': 'admin',
        'password': 'password',
        'database': 'voip_metrics'
    }

    mongodb_config = {
        'connection_string': 'mongodb://mongodb:27017/',
        'database': 'voip_analytics',
        'problems_collection': 'problems'
    }

    detector = VoipAnomalyDetector(kafka_config, influxdb_config,
mongodb_config)
    detector.run()

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмне забезпечення для оцінки сервісів IP-телефонії.

Виконав студент групи КНТ-121
Керівник, к.т.н, доцент

Ілля ІСМАІЛОВ
Степан СКРУПСЬКИЙ

Рисунок В.1 – Слайд 1

Мета та задачі

Об'єкт дослідження: процес розроблення програмного забезпечення для моніторингу та аналізу ефективності сервісів IP-телефонії.

Предмет дослідження: методи та програмні засоби автоматизованої оцінки якості сервісів IP-телефонії на основі множини метрик продуктивності та надійності.

Мета роботи: підвищення точності та оперативності оцінювання сервісів IP-телефонії шляхом розроблення спеціалізованого програмного забезпечення з комплексною системою метрик та алгоритмів аналізу.

Завдання дослідження:

- проаналізувати існуючі методи та програмні рішення для оцінки якості IP-телефонії, визначити їх переваги та недоліки;
- розробити систему метрик для комплексної оцінки сервісів;
- спроектувати архітектуру програмного забезпечення для збору, обробки та аналізу даних про функціонування сервісів IP-телефонії;
- реалізувати алгоритми обчислення інтегрального показника якості на основі зібраних метрик з використанням методів машинного навчання;
- розробити інтерфейс користувача для візуалізації результатів оцінювання та підтримки прийняття рішень;
- провести тестування розробленого програмного забезпечення.

Рисунок В.2 – Слайд 2

Порівняння систем моніторингу VoIP

Функціональність	SolarWinds VoIP & Network Quality Manager	PRTG Network Monitor	ManageEngine VoIP Monitor	Observer Platform
Моніторинг якості дзвінків	Так	Так	Так	Так
Аналіз сигнального трафіку	Так	Обмежений	Так	Так
Аналіз медіатрафіку	Так	Так	Так	Так
Синтетичні тести	Так	Ні	Обмежені	Так
Аналіз CDR	Так	Ні	Так	Обмежений
Карта мережі	Так	Так	Так	Так
Оповіщення	Так	Так	Так	Так
Звіти	Розширені	Базові	Розширені	Розширені
Інтеграція з ITSM	Так	Обмежена	Так	Обмежена
Прогностична аналітика	Обмежена	Ні	Обмежена	Так
Ціна	Висока	Середня	Середня	Висока

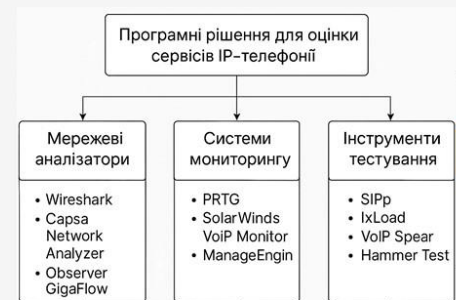


Рисунок В.3 – Слайд 3

Архітектура типової системи моніторингу IP-телефонії

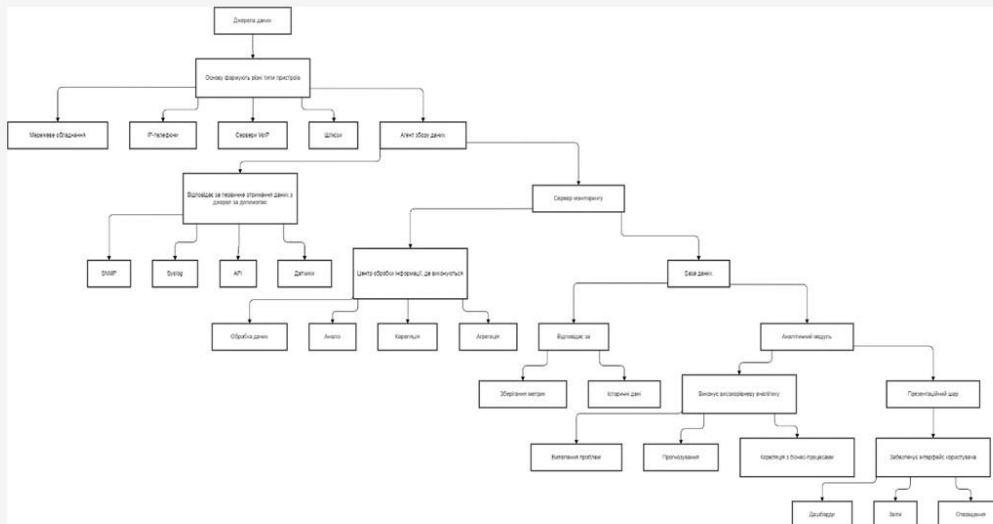


Рисунок В.4 – Слайд 4

Модель потоків даних у системі оцінки сервісів IP-телефонії

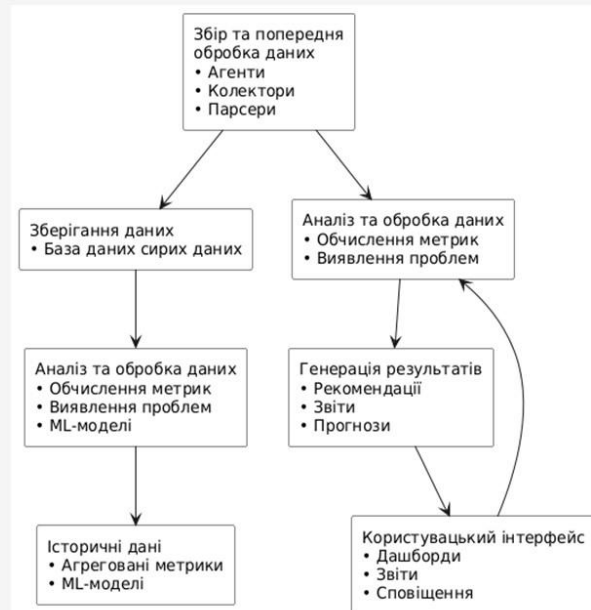


Рисунок В.5 – Слайд 5

Високорівнева архітектура системи оцінки сервісів IP-телефонії

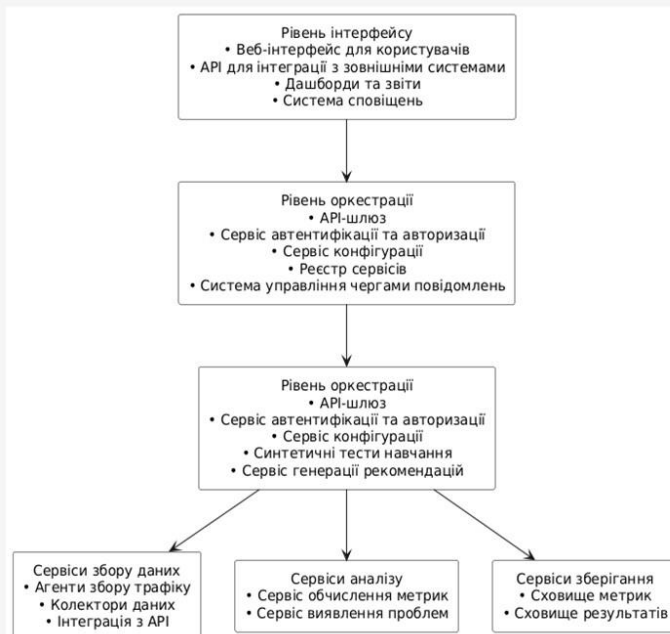


Рисунок В.6 – Слайд 6

Вибір технологій для реалізації різних рівнів системи

Рівень системи	Обрана технологія	Обґрунтування вибору
Рівень інтерфейсу	React	Висока продуктивність, компонентний підхід, велика спільнота розробників, можливість створення інтерактивних дашбордів
Рівень оркестрації	Kubernetes	Гнучка система оркестрації контейнерів, можливість автоматичного масштабування, самовідновлення та розподіленого зберігання стану
API-шлюз	Kong API Gateway	Відкритий вихідний код, підтримка балансування навантаження, маршрутизації, кешування та автентифікації
Сервіс автентифікації	OAuth 2.0 з Keycloak	Стандартизований протокол автентифікації, підтримка SSO, гнучке управління доступом
Сервіси збору даних	Go (для агентів), Node.js (для колекторів)	Go забезпечує високу продуктивність і низьке споживання ресурсів для агентів, Node.js - гнучкість для інтеграцій
Сервіси аналізу	Python з TensorFlow і scikit-learn	Потужні бібліотеки для машинного навчання та аналізу даних, простота реалізації алгоритмів
Сховище метрик	InfluxDB	Висока продуктивність для запису часових рядів, підтримка SQL-подібної мови запитів, гнучка схема даних
Сховище конфігурацій	etcd	Розподілене сховище ключ-значення з підтримкою консенсусу, інтеграція з Kubernetes
Сховище результатів аналізу	MongoDB	Гнучка схема даних, підтримка складних запитів, масштабованість
Система черг повідомлень	Apache Kafka	Висока пропускна здатність, підтримка потокової обробки, розподіленість
Система візуалізації	Grafana	Розширені можливості для створення дашбордів, підтримка різних джерел даних, система сповіщень
CI/CD pipeline	GitLab CI	Інтеграція з системою контролю версій, гнучкі налаштування, підтримка контейнеризації

Рисунок В.7 – Слайд 7

Схема
колекцій
бази даних
MongoDB
для
зберігання
результатів
аналізу

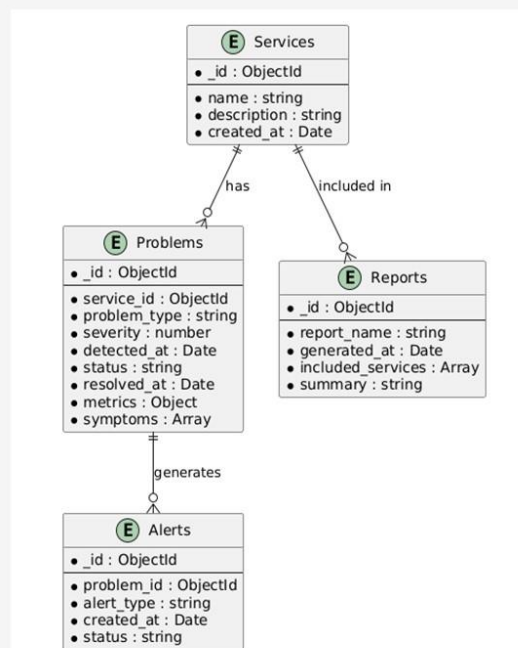


Рисунок В.8 – Слайд 8

Загальна
структура
інтерфейсу
користувача
програмного
забезпечення

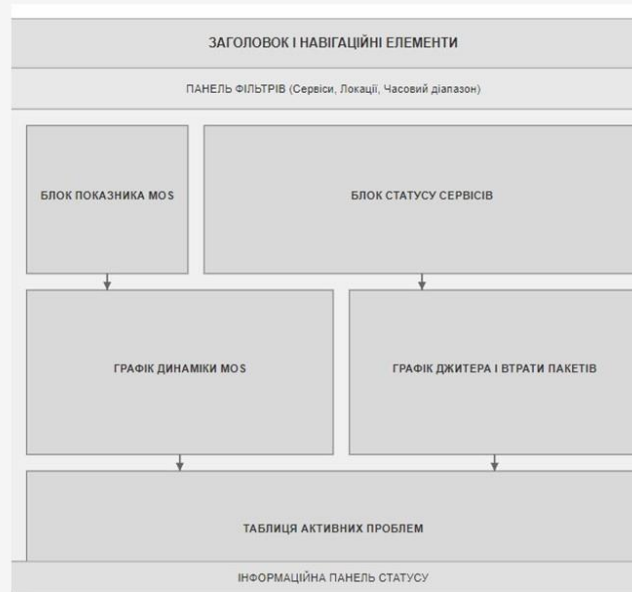


Рисунок В.9 – Слайд 9

Інтерфейс
користувача
програмного
забезпечення

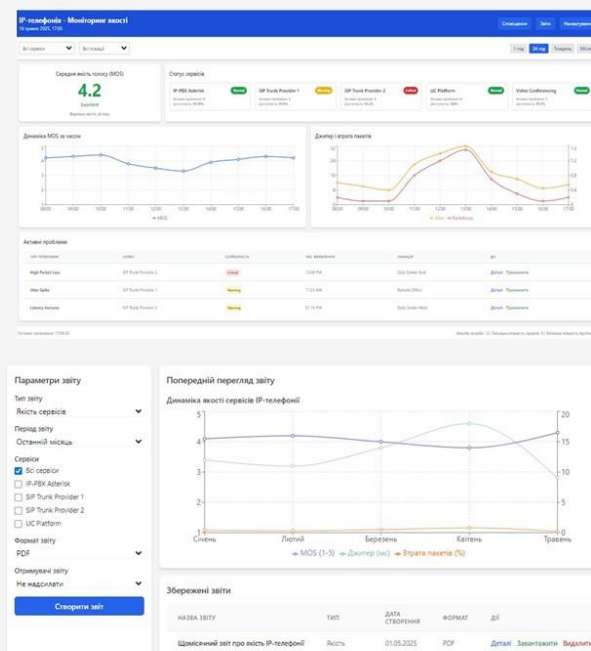


Рисунок В.10 – Слайд 10

Висновки

В ході виконання дипломної кваліфікаційної роботи розроблено програмне забезпечення для оцінки сервісів IP-телефонії, що підвищує точність та оперативність оцінювання якості VoIP-систем шляхом комплексного аналізу різних аспектів їх функціонування.

Проведено аналіз технологій IP-телефонії та існуючих програмних рішень. Визначено, що для комплексної оцінки необхідно враховувати мережеві характеристики, якість голосу, надійність сервісу та функціональність. Встановлено, що існуючі рішення часто фокусуються на окремих аспектах оцінки, мають обмежені можливості для порівняльного аналізу та недостатньо використовують методи машинного навчання.

Спроектровано мікросервісну архітектуру системи з чотирма основними рівнями: інтерфейс, оркестрація, сервіси та сховища даних. Для реалізації компонентів обрано технології React, Kubernetes, InfluxDB та Python з бібліотеками машинного навчання.

Розроблено алгоритми оцінки сервісів IP-телефонії, включаючи обчислення MOS на основі E-моделі, виявлення аномалій та прогнозування якості. Запропоновано інтегральний показник для комплексної оцінки якості з налаштуванням вагових коефіцієнтів.

Створено ефективну структуру баз даних для зберігання метрик в InfluxDB та результатів аналізу в MongoDB.

Розроблено інтерфейс користувача за принципами User-Centered Design для забезпечення зручності використання.

Тестування підтвердило високу точність обчислення метрик, стійкість системи до часткових відмов, здатність обробляти значні обсяги даних та відповідність вимогам безпеки.

Практична значущість роботи полягає у створенні інструментарію для адміністраторів мереж та IT-менеджерів, що дозволяє об'єктивно оцінювати якість сервісів IP-телефонії, виявляти проблеми та оптимізувати конфігурації систем.

Рисунок В.11 – Слайд 11