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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ”

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

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

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

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

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

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

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

Язловецькому Богдану Максимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ*

2. Керівник роботи *Петренюк Володимир Ілліч,канд. фіз.-мат. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

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

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ*

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

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

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

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

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Петренюк В.І.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Язловецький Б.М.
(прізвище та ініціали)

АНОТАЦІЯ

Язловецький Б.М. Програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

Метою розробки є програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

Результат роботи – програмна реалізація системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

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

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

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

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, SaaS, віртуальна АТМ

ABSTRACT

Yazlovetskyi B.M. Software for the SaaS cloud service system for implementing a virtual ATM. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for the SaaS cloud service system for implementing a virtual ATM.

The purpose of the development is software for the SaaS cloud service system for implementing a virtual ATM.

The result of the work is software implementation of the SaaS cloud service system for implementing a virtual ATM.

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

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

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

The program was developed in the Python environment.

Keywords: computer engineering, SaaS, virtual ATM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БД – база даних
КМЗ – корпоративна мережа зв’язку
ЛОМ – локальна обчислювальна мережа
ОЗП – оперативно-запам’ятовувальний пристрій
ПАТМ – пристрій автоматизованої телефонної мережі
ПЗ – програмне забезпечення
ПП – програмний продукт
СПД – система передачі даних
СУБД – система управління БД
ТфОП – телефонний оператор
CNG – Comfort Noise Generator. Генератор комфортного шуму
DSP – Digital Signal Processor. Процесор цифрової обробки сигналів
DTMF – Dual Tone Multi-Freqency. Багаточастотна система кодування цифр номера
GK – Gatekeeper. Воротар . Виконує функції керування зоною мережі H.323
GW – Gateway. Шлюз. Апаратно-програмний комплекс, що забезпечує обмін даними між мережами різних типів
H.323 – Рекомендація ITU-T, що визначає системи мультимедійного зв'язку в мережах з пакетною комутацією
H.248 – Протокол керування транспортним шлюзом
IP – Internet Protocol. Протокол міжмережної взаємодії
LPC Linear Prediction Coding. Кодування з лінійним пророкуванням
MCU – Multipoint Control Unit. Пристрій керування конференцією
MG – Media Gateway. Транспортний шлюз
MGCP – Media Gateway Control Protocol. Протокол керування шлюзами

MP – Multipoint processor. Процесор для обробки інформації користувачів при централізованих конференціях

OSI – Open System Interconnection. Взаємодія відкритих систем

PPP – Point-to-Point Protocol. Протокол двостороннього зв'язку

RADIUS – Remote Authentication Dial-In User Service. Протокол автентифікації й авторизації абонентів, а також обліку обсягу наданих їм послуг

RAS – Registration Admission and Status. Протокол взаємодії термінального встаткування з gatekeeper. Входить у сімейство протоколів H.323

RSVP – Resource Reservation Protocol. Протокол резервування ресурсів

RTCP – Real-time Transport Control Protocol. Протокол контролю транспортування інформації в реальному часі

SIP – Session Initiation Protocol. Протокол ініціювання сеансів зв'язку

TAPI – Telephony Applications Programming Interface. Інтерфейс для програмування телефонних додатків

TCP – Transmission Control Protocol. Протокол керування передачею (даних) Основний транспортний протокол у стеці протоколів TCP/IP. Встановлює зв'язок між двома ПК й організує обмін даними в дуплексному режимі

TCP/IP – Transmission Control Protocol/Internet Protocol. Стек протоколів, що забезпечують організацію зв'язку між комп'ютерами в мережі Інтернет

UDP – User Datagram Protocol. Протокол передачі дейтаграмм користувача. Подібно TCP, використовує для доставки даних протокол IP. На відміну від TCP/IP, передбачає обмін дейтаграммами без підтвердження

VoIP – Voice over Internet Protocol. Технологія, що дозволяє використовувати IP-мережу для передачі мовної інформації

ВСТУП

Актуальність теми. Віртуальні АТМ – один із тих напрямків хмарних сервісів для бізнесу, які дуже швидко розвиваються. Вони не тільки дозволяють знизити капітальні витрати й операційні витрати, але й допомагають підвищити продуктивність роботи, одержати об'єктивні дані для оцінки її ефективності й прийняття управлінських рішень.

Коли говорять про хмарну телефонію, то по суті мова йде про віртуальну, або, як їх ще називають, хмарну, автоматизовану телефонну мережу (АТМ) – хмарний сервіс SaaS для побудови корпоративної системи телефонії. Через наданий Web-інтерфейс клієнт може підключати до системи IP-телефони й софтфони, налаштовувати алгоритми обробки викликів, задіяти додаткові додатки й сервіси, переглядати різноманітну статистику й навіть одержувати аналітичні звіти по викликах. Для здійснення дзвінків на стаціонарні й мобільні телефони використовуються сервіси операторів IP-телефонії.

Телефонія була й залишається одним з основних комунікаційних засобів як усередині компанії, так і при взаємодії із зовнішніми клієнтами. Як і у випадку інших корпоративних сервісів, перехід на хмарну модель використання має цілий ряд переваг. Віднесемо до них зниження капітальних витрат, оплату сервісу в міру споживання, швидке уведення в експлуатацію (як правило, за один день), необмежене масштабування й надійність сервісу. Явним підтвердженням зручності хмарних АТМ є щорічний ріст числа їхніх абонентів.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

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

– Огляд існуючих систем хмарного сервісу SaaS для реалізації віртуальної АТМ.

– Дослідження системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

– Програмна реалізація системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі хмарного сервісу SaaS для реалізації віртуальної АТМ.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Переваги хмарних АТМ

Скорочення капітальних/операційних витрат і витрат на зв'язок

Подорожчання закордонного встаткування й зменшення IT-бюджетів змушують компанії звернути увагу на хмарні рішення, які дозволяють обійтися без покупки офісної АТМ і заощаджувати на міжнародних і міжміських дзвінках. Перевага хмарних АТМ із фінансової точки зору – зниження капітальних витрат при впровадженні й простої прогнозування операційних витрат.

Капітальні інвестиції в більшості випадків будуть складатися з вартості телефонних апаратів і підключення телефонних номерів. Компанії не потрібно придбати у власність дорогі телефонні станції, сервери й ліцензії на програмне забезпечення. Операційні витрати також знижуються, оскільки немає необхідності в обслуговуванні й підтримці власної телефонної інфраструктури.

Масштабованість

Клієнт може замовити будь-який обсяг ресурсів – рішення для десятка абонентів або для тисяч співробітників. Можливо як збільшення числа користувачів, номерів, ліній, так і їхнє скорочення. Вартість, як правило, прив'язана до кількості підтримуваних користувачів і функціональних можливостей.

Надійність

Нерідко основним фактором на користь переносу телефонії в хмару стає відказостійкість. Провайдер, що надає послугу, як правило, дає фінансову гарантію безвідмовної роботи сервісу.

Функціональність

Крім функцій телефонії, АТМ надають інформаційно-аналітичні сервіси, що дозволяють виміряти й поліпшити показники обслуговування клієнтів по телефоні, включаючи кількість прийнятих і пропущених викликів, час очікування відповіді на лінії, середню тривалість розмови, розподіл навантаження по годинниках і ряд інших. За рахунок поліпшення даних показників можна підвищити лояльність клієнтів і обсяги продажів. Запис дзвінків, історія переговорів, статистика й інші функції АТМ допомагають керівникам ефективно управляти своїми співробітниками, що в остаточному підсумку сприятливо відбивається на якості роботи із клієнтами.

1.2 Область застосування

Областю застосування є хмарний сервіс SaaS. SaaS (Software as a Service) – це модель використання бізнес-додатків у форматі Інтернет-сервісів.

SaaS додатка працюють на сервері SaaS-провайдеру, а користувачі одержують до них доступ через Інтернет-браузер. Користувач не купує SaaS-Додаток, а орендує його – платить за його використання деяку суму на місяць. У такий спосіб досягається економічний ефект, що вважається одним з головних переваг SaaS.

SaaS провайдер піклується про працездатність додатка, здійснює технічну підтримку користувачів, самостійно встановлює відновлення. Таким чином, користувач менше думає про технічну сторону питання, а зосереджує на своїх бізнес-цілях.

Основні переваги SaaS над традиційним програмним забезпеченням:

– більш низька вартість володіння;

– більш короткі строки впровадження;

– низький поріг входу (можна швидко й безкоштовно протестувати);

– завдання по підтримці й відновленню системи повністю лягають на плечі SaaS-провайдеру;

– повна мобільність користувача, обмежена лише " Інтернет-покриттям";

– підтримка географічно розподілених компаній і вилучених співробітників;

– низькі вимоги до потужності комп'ютера користувача;

– кроссплатформеність.

Недоліками SaaS вважаються небезпека передачі комерційних даних сторонньому провайдеру, невисока швидкодія й ненадійність доступу через перебої з Інтернетом. Однак імідж, що зміцнює, SaaS-провайдерів, розвиток технологій шифрування й широкополосного доступу в Інтернет поступово розсіюють ці страхи.

Альтернативи SaaS

Через перераховані вище страхи з'явилися альтернативні технології стосовно SaaS. Вони являють собою проміжні варіанти переходу від традиційного ПЗ до SaaS, і швидше за все, незабаром зникнуть.

S+S. Це альтернативний бренд, що просувається Microsoft, що відрізняється від SaaS тим, що на комп'ютері користувача використовується не браузер, а програмний клієнт.

Оренда (хостинг) додатків. Цей варіант відрізняється від SaaS лише архітектурою серверної частини й не помітний для користувача. Тому часто хостери додатків називають свої послуги SaaS-сервісами. Відмінність у тому, що класичні SaaS сервіси мають multitenant-архітектуру, тобто один додаток обслуговує багато клієнтів, а хостинг додатків припускає установку окремої копії для кожного клієнта. Другий варіант дає більше можливостей налаштування, але в той же час, він більше складний для адміністрування й відновлення, і тому коштує дорожче.

Використання хмарних платформ. Компанії, які бояться віддавати свої дані сторонньому провайдеру, іноді обмежуються тим, що орендують в Інтернеті

не додатка, а тільки комп'ютерні потужності й установлюють на них свої (куплені) системи. Для такого варіанта існують хмарні платформи.

Хоча для багатьох керівників технологія SaaS (програмне забезпечення як сервіс) усе ще здається справою майбутнього, уже можна точно сказати, що буде після її. Ми не претендуємо на те, щоб точно вгадати назву нової технології. Назвемо її, приміром, SoIP (Service over IP = сервіс через Інтернет). Але справа не в назві, а в змісті. Зміст у тому, що реально бізнесу не потрібно програмне забезпечення (ні на власному сервері, ні на сервері стороннього провайдеру). Їм потрібні рішення конкретних завдань. Їм потрібні сервіси, які вони можуть замовляти через Інтернет, контролювати й одержувати готові результати через Інтернет. Звичайно, це не якась супер-нова ідея. SaaS-провайдери, спілкуючись зі своїми клієнтами самі приходять до розуміння цього й додають у свої додатки непрограмну цінність

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Віртуальна АТМ – яку вибрати

Багато сучасних компаній не представляють своєї роботи без IP-телефонії, що дозволяє справлятися з бізнес-завданнями значно швидше й дешевше, ніж традиційні телефонні мережі.

Переваг в Інтернет-телефонії безліч:

- конференц-зв'язок;
- переадресація дзвінків;
- відсутність фізичної прив'язки;
- відеозв'язок;
- і багато чого іншого.

Однак вибрати VOIP-провайдеру досить непросто. Зараз у Росії працює більше чотирьох сотень провайдерів, кожний з яких переконує підключитися саме до нього. Як грамотно підступитися до цієї армади й зрозуміти, яку віртуальну АТМ вибрати?

Перш ніж підписувати договір і оплачувати послуги, звичайно ж, потрібно уважно придивитися до пропонованих умов. Але в першу чергу, варто визначитися зі своїми потребами, скласти список очікувань від IP-телефонії, і вже відповідно до цього, займатися пошуком. Очевидно, що кожний з операторів має сильні й слабкі сторони. Тільки правильно розставивши пріоритети, можна одержати добре працюючий канал зв'язку із клієнтами, на оптимальних умовах. Говорячи про VOIP-провайдери, можна використовувати нескінченну безліч критеріїв для порівняння, але існує список основних пунктів, з огляду на які,

значно простіше знайти постачальника послуг, відмінно підходящої конкретної компанії.

Популярність

Варто відзначити, що перші компанії, які розвивали Інтернет-телефонію, були засновані ще 20-25 років тому. За цей час ринок розрісся до величезних масштабів, а компанії-засновники одержали гігантську популярність. І при необхідності підключення IP-телефонії, у число претендентів звичайно попадають саме 5-7 самих великих операторів. Такий підхід легко пояснити устояною думкою, мол, популярний – значить надійний і чесний. Подібне твердження можна назвати вірним, але разом з тим важливо підкреслити, що на сьогоднішній день ризик нарватися на відвертих аферистів у даній сфері практично відсутній.

Це стосується навіть що тільки з'явилися провайдерів, не говорячи вуж про просто порівняно невеликий VOIP-операторів, що успішно працюють на ринку, але не одержали божевільної популярності. І в цьому випадку висока популярність є дуже неоднозначним показником. За рахунок великої кількості клієнтів, великі провайдери змушені працювати в режимі конвеєра. Як підсумок, значимість окремо взятого замовника знижується, падає рівень турботи й зацікавленості.

Маленькі компанії, з не таким більшим колом клієнтів, найчастіше виявляються набагато більше гнучкими, чуйними й уважними. Вони дорожать кожним замовником і приділяють рішенню питань, що турбують, стільки часу, скільки буде потрібно. Тому далеко не завжди вигідно й правильно виконувати вибір sip-провайдеру із числа самих великих і розкручених компаній.

Якість зв'язку

На жаль, похвастатися високою якістю телефонного зв'язку можуть не всі постачальники подібних послуг. Але ж у бізнесі вимоги до якості підвищені: не поговорив – втратив клієнта й гроші. Якщо розмова буде супроводжуватися ефектом луни, затримками в передачі голосу, його перекручуванням, шумами й

іншими ріжуче вухо перешкодами, спілкуватися в таких умовах буде не занадто комфортно. Найімовірніше, потенційний клієнт просто перерве бесіду. Зв'язано такі проблеми можуть бути як з невірним налаштуванням, або некоректною побудовою мережі провайдеру, так і з низькою пропускною здатністю його каналів зв'язку, застосовуваних для передачі голосових повідомлень абонентів.

В ідеалі, звернувши увагу на яку-небудь компанію, непогано було б вивчити відгуки про якість зв'язку цього провайдеру. В Інтернеті без праці можна знайти таку інформації. Реальні коментарі, особливо негативні, звичайно цілком обґрунтовані й аргументовані. Тому якщо користувачі характеризують копанню як погану, те її можна сміло відмітати. У випадку знаходження похвали й подяк, варто перейти до наступного етапу – спробувати особисто поспілкуватися з людьми, що користуються послугами даного оператора, і довідатися, чи задоволені вони якістю надаваних послуг.

Технічна підтримка

Інтернет-телефонія, на відміну від традиційного мідного кабелю, являє собою вкрай складну технологію. І навіть краща віртуальна АТМ часом може давати збій. Між що дзвонить і оператором перебуває безліч ланок, від справності й вірного налаштування яких залежить працездатність каналу зв'язку і якість кожної розмови. При виникненні неполадок, саме техпідтримка повинна займатися консультуванням клієнтів і повідомленням відповідних фахівців про наявність проблем. Дуже важливо довідатися, протягом якого часу звичайно усуваються збої, уточнити графіка роботи підтримки, по можливості переконатися у кваліфікації співробітників.

Якщо обслуговування клієнтів поставлене на потік, і щоб додзвонитися до оператора доводиться витратити більше 30 хвилин – не варто зв'язуватися з таким провайдером. Принаймні, розраховувати на оперативну технічну й консультаційну підтримку вже точно не прийдеться. Також забути про сучасну допомогу можна, якщо служба технічної підтримки працює не цілодобово.

Тарифи

До рекламних заяв VoIP-операторів, як і до всякої іншої реклами, варто підходити гранично обережно. Провайдери спритно жонглюють більшою кількістю значень, і нерідко при ближчому розгляді привабливих цін, виявляється, що це нічні тарифи, вартість без обліку податків, тарифи вихідного дня або інші хитрості.

Собівартість телефонного трафіку для компаній зараз практично ідентична, і розкид цінових пропозицій залежить більшою мірою безпосередньо від націнки оператора. Як правило, якщо тарифи істотно нижче, ніж у конкурентів, це свідчать про повсюдну економію, що чревате жахливою якістю послуг. Або ж це говорить про те, що в пропозиції закладені навмисно сховані платежі.

Щоб вибрати віртуальну АТМ із оптимальним співвідношенням ціни і якості, необхідно озброїтися калькулятором і витратити певні зусилля, щоб з'ясувати, скільки ж насправді прийде платити за необхідний набір послуг.

Функціонал

Вивчивши сайти підібраних VoIP-провайдерів, необхідно докладніше ознайомитися з функціоналом віртуального номера. Особлива увага варто приділити частині надаваних додаткових послуг, і сформувати свій погляд на те, які із цих сервісів для компанії корисні або навіть критично необхідні, а які – зайва витрата грошей. Чималий оператор не буде нав'язувати непотрібні опції за окрему плату, а от потенційні можливості функціонала варто враховувати. У випадку розширення бізнесу, цілком можуть знадобитися нові функції. Виникне безліч незручностей, якщо їхнє підключення буде неможливо без зміни вже звичного оператора.

Отже, підіб'ємо короткий підсумок:

1. Самий популярний провайдер – не завжди кращий для Вас.

2. Кращий спосіб переконатися як послуги – поспілкуватися з реальними діючими абонентами провайдеру.

3. Погана технічна підтримка гарантує вам проблеми й збитки в майбутньому.

4. Якщо провайдер іде на виверти й намагається сховати від вас реальну вартість послуг – не коштує з ним зв'язуватися.

5. Не потрібно платити за зайвого функціонала, але завжди повинна бути можливість розширити набір використовуваних інструментів.

Найпоширенішою операційною системою є Windows. Тому SIP-клієнтів під цю операційну системи досить багато. У список самих популярних клієнтів ми внесемо наступні програми:

– 3CX VOIP Phone.
– IP-Phone.
– X-Lite.
– Zoiper.
– QIP Infium.

Розглянемо їх більш докладно.

3CX VOIP Phone

Програмний телефон 3CX VOIP Phone для Windows є одним із самих популярних SIP-клієнтів. Перевагою програми є невелика вага, що забезпечує швидке завантаження на будь-якому каналі. Після запуску на екрані з'явиться приємний інтерфейс, що нагадує самий звичайний мобільний телефон. При першому старті програма запропонує додати аккаунт.

3CX VOIP Phone підтримує відразу декілька аккаунтів, що придасться тим, хто користується послугами відразу декількох провайдерів.

Якщо ви скачали 3CX VOIP Phone з офіційного сайту, варто додати українську мову – це робиться в налаштуваннях, які викликаються за допомогою середньої кнопки (як на смартфонах з Android). Який функціонал цієї програми:

– Можливість здійснення відеовикликів.

– Установка статусу – доступний, немає на місці, автовідповідь, переадресація на голосову пошту.

– Можливість здійснення дзвінків відразу по декількох лініях.

– Перегляд історії викликів.

– Запис розмов у файл.

– Підтримка швидкого набору.

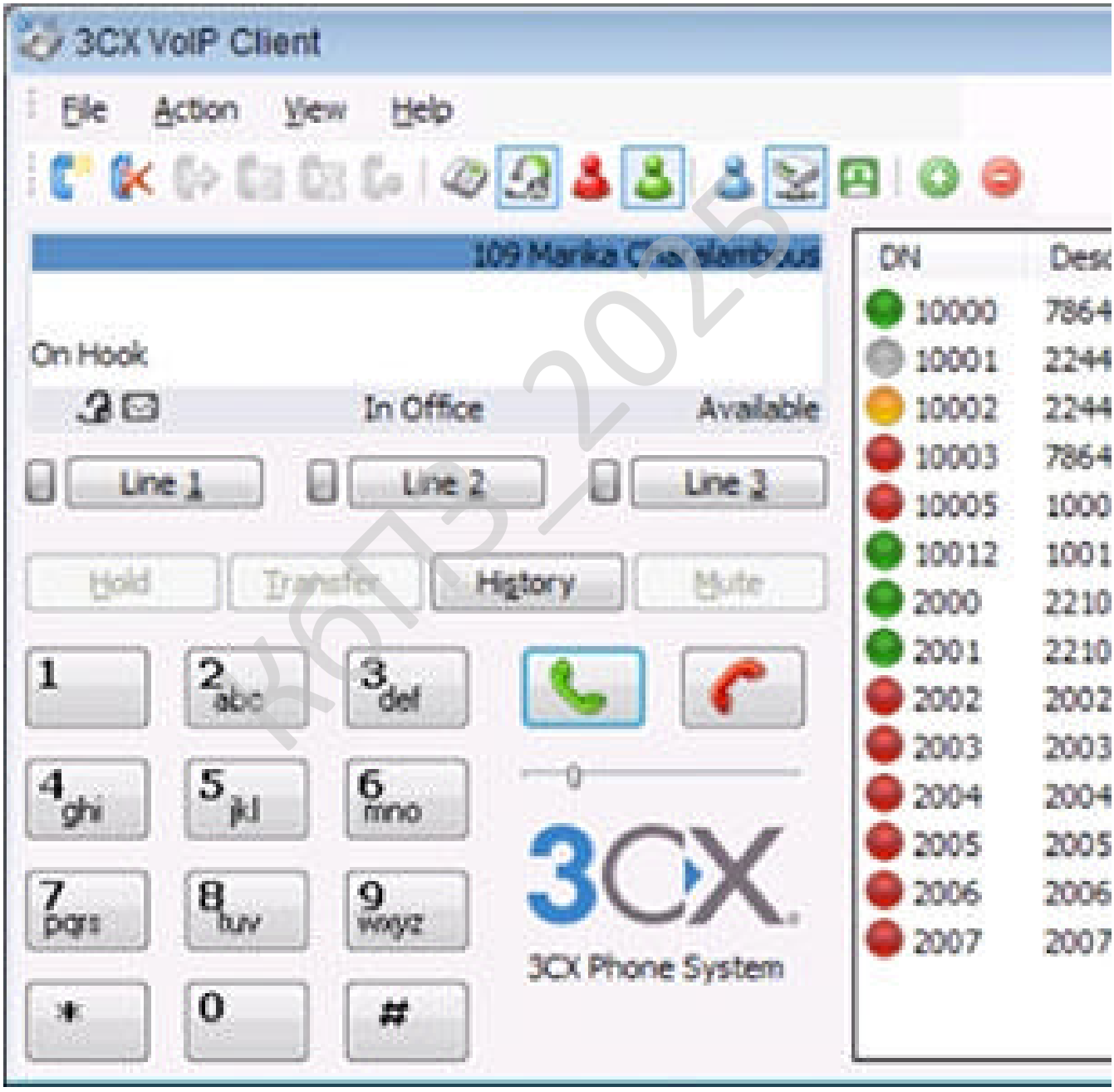

Рисунок 2.1 – Інтерфейс користувача 3CX VOIP Phone

Програма досить проста, але кирилізація в неї страждає – деякі пункти як і раніше залишаються на англійському. Зайшовши в налаштування ми виявимо, що їх дуже мало. Користувачі можуть вибрати звукові пристрої й камеру, визначити поводження програми при знаходженні комп'ютера в різних режимах, задати придушення шумів. Тут же задається зовнішній вигляд програми – є підтримка різних скинів.

IP-Phone

Програма IP-Phone є найпростішим і зрозумілим програмним IP-телефоном. Можна сказати, що це найпростіший SIP-клієнт для Windows. Сама програма українськомовна, хоча деякі пункти на англійському. Безсумнівним перевагом є те, що цей телефон уміє запитувати й відображати баланс на використовуваному аккаунті. Також підтримується відображення номерів доступу.

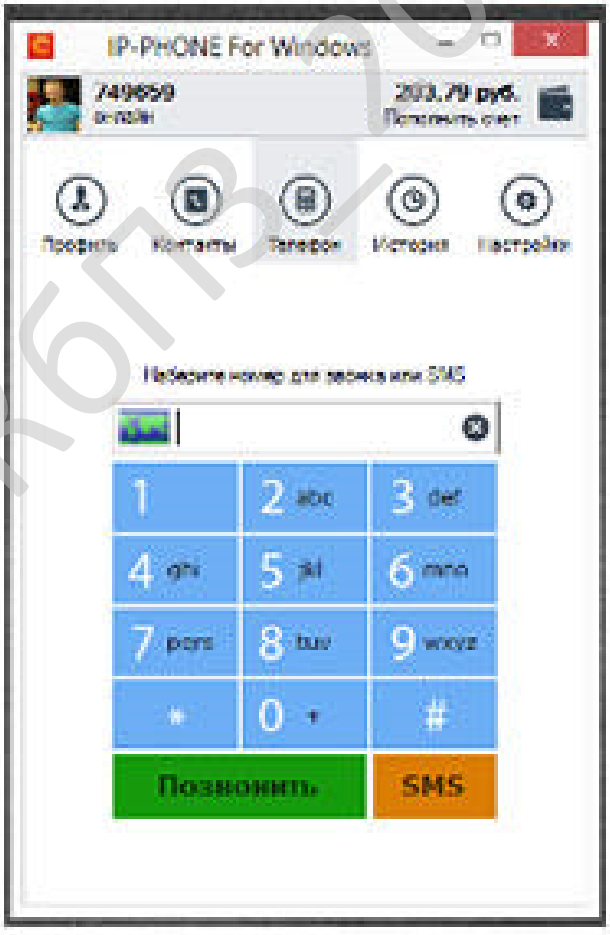

Рисунок 2.2 – Інтерфейс користувача IP-Phone

Варто сказати, що програма орієнтована на SIP-провайдеру Comtube і працює тільки з ним. Підтримується тільки один аккаунт.

При налаштуванні даної програми вказуються логін і пароль від аккаунта на сайті Comtube. Авторизація по SIP ID не підтримується.

X-Lite

Програмний телефон X-Lite має англомовний інтерфейс і досить приємним зовнішнім виглядом.

Рисунок 2.3 – Інтерфейс користувача X-Lite

Прямо на головному екрані розташовуються великі цифрові кнопки – промахнутися повз них не вийде. Нижче вибираються контакти й вибрані номери, відображається історія викликів. Ледве вище виставляється статус, там

же викликається віконце для обміну текстовими повідомленнями. Тут же розташовуються регулятор гучності й рядок пошуку контактів.

У налаштуваннях програми практично нічого немає – отут редагуються звукові оповіщення, вибираються звукові пристрій, задаються кодеки для роботи з тими або іншими провайдерами. Тут же вибирається мова, якому потрібно скачати й установити окремо. Налаштування аккаунта (причому одного) розташовуються в окремій вкладці, тут можна виконати тонке налаштування.

Zoiper

SIP-клієнт Zoiper випускається в трьох видах:

– Безкоштовна версія.

– Бізнес-Версія.

– Індивідуальна версія.

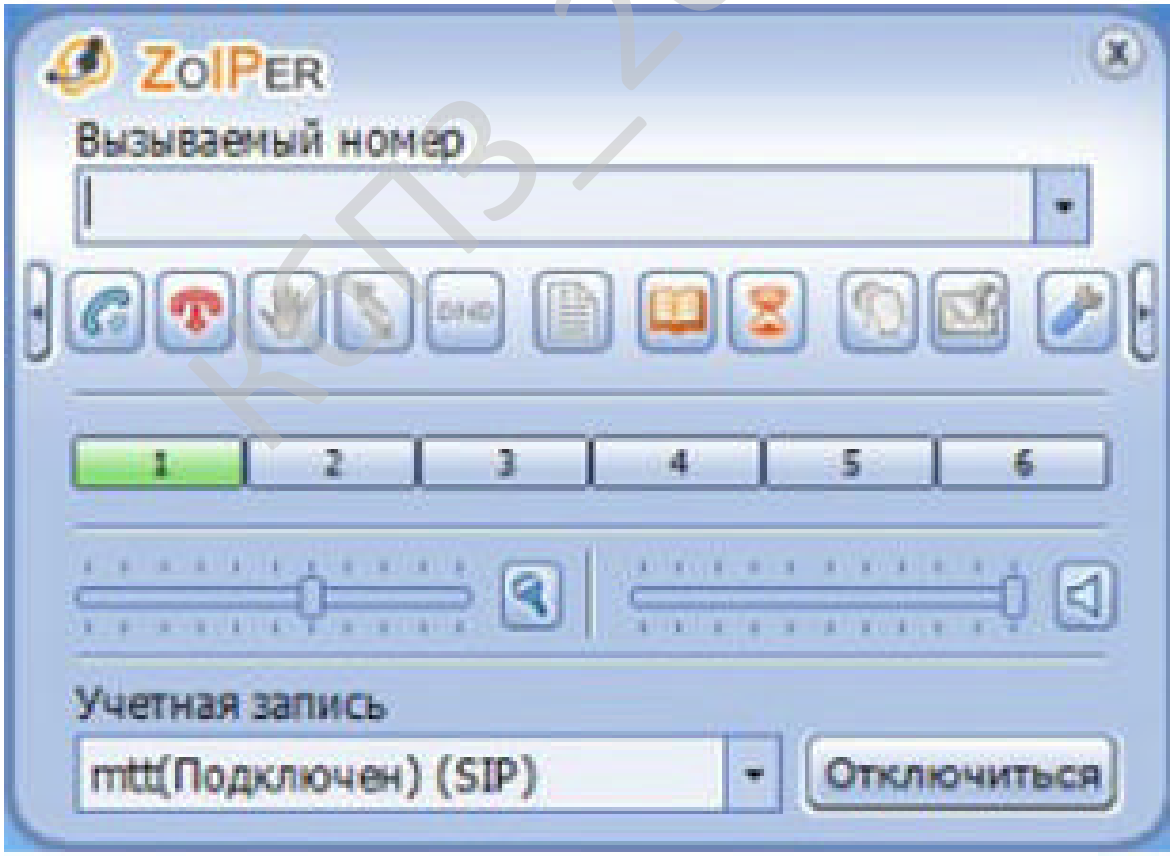

Рисунок 2.4 – Інтерфейс користувача Zoiper

Безкоштовна версія клієнта доступна для всіх користувачів, але вона декілька обмежена в можливостях. У бізнес-версії присутні підтримка додаткових алгоритмів стиску звуку, поліпшена й розширена робота з контактами, є інтеграція в браузер, функція пересилання файлів і конференц-зв'язок. Вартість програми становить 33 євро. Індивідуальна версія платна, тут виконується брендування під замовника, можливе замовлення додаткових опцій і індивідуального дизайну, відображається баланс. Вартість цієї версії підраховується індивідуально.

Безкоштовна версія Zoiper підтримує протоколи SIP, IAX і XMPP, уміє працювати з декількома аккаунтами. Контакти зберігаються локально або на серверах Zoiper, у глобальній адресній книзі з пошуком. Українська мова вже включена в стартовий дистрибутив. Налаштування дуже й дуже широкі, програму можна налаштувати по своєму смаку. Функціонал програми:

– Підтримка відеовикликів.
– Генерація комфортного шуму.
– Шумозаглушення.
– Підтримка скинів.

Переадресації в безкоштовній версії ні, як немає й розширеної роботи з контактами. Інтерфейс не дуже зручний і не дуже гарний**.** Також занадто нав'язливо пропонується придбати бізнес-версію – позначка про це зустрічається буквально на кожному кроці.

Модуль в QIP Infium

У популярному мультипротокольному мессенджері QIP Infium є присутнім модуль для роботи з SIP-телефонією. Налаштування зводиться до прописування даних аккаунта й вибору необхідних кодеків. В інтерфейсі присутній простенька панелька з дозвоном, є можливість підключення USB-телефону. Також реалізована можливість швидкого доступу до сервісів провайдеру, але працює вона не завжди – імовірно, все залежить від налаштувань.

Рисунок 2.5 – Інтерфейс користувача QIP Infium

Ця «дзвонилка» підійде тим, хто не хоче захаращувати оперативну пам'ять додатковими додатками. Але потрібно пам'ятати, що ніякого додаткового функціонала (крім придушення луни) тут немає.

Клієнти для інших ОС

Користуватися IP-телефонією можна не тільки на комп'ютері, але й на смартфоні. Оптимальний SIP-клієнт для Android – це 3CXPhone – VoIP/SIP phone, його можна скачати через Google Play. Програма дуже функціональна, відрізняється швидкістю роботи й приємним інтерфейсом. Недоліком є відсутність української мови.

Якщо ви користуєтеся сервісом Comtube, ми рекомендуємо вам скачати з Google Play додаток IP-Phone для Android. Програма відрізняється легкістю в налаштуваннях, уміє показувати вартість дзвінків, може заміняти собою стандартну звонилку. Це ідеальний клієнт для Comtube, що не завантажує смартфон і не приводить до швидкої витрати батареї. Інтерфейс українськомовний, передбачений показ сервісної інформації – баланс, тариф, дані користувача.

Самий популярний SIP-клієнт для Iphone – це програма 3CXPhone. Вона вміє працювати з декількома обліковими записами й має велику гнучкість у налаштуваннях, як і її побратим для Windows. Не дивно, що цей SIP-клієнт став самим популярним. Попитом користується й клієнт ForFone – програма має зрозумілий інтерфейс, дозволяє спілкуватися голосом і в режимі чата. У число недоліків входить те, що вона орієнтована на роботу з однойменним SIP-провайдером.

Абоненти Comtube зможуть скористатися додатками від свого провайдеру – їх можна знайти в AppStore. Для здійснення дзвінків необхідна програма IP-Phone, аналогічна програмам для вищезгаданих платформ.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

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

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо. Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Python абсолютно безкоштовний.

Швидкість виконання коду Python

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

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

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

Використання Python

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

Застосування мови програмування Python:

1. BitTorrent – протокол для обміну даними.

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

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

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

5. World of Tanks.

6. Вільна енциклопедія Вікіпедія.

7. Пошукова система Google.

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

9. YouTube – популярне відеосховище.

Версії Python

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

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

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

Інструкція машинної мови існує для кожної операції, яку процесор здатний виконати – є інструкція для додавання чисел, є інструкція для віднімання

чисел і т.д. Увесь набір інструкцій, який центральний процесор може виконати, відомий як набір інструкцій процесора.

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

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

Отримати

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

Декодувати

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

Виконати

Останній крок циклу – виконати операцію.

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

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

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

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

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

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

1. Компілятор.

2. Інтерпретатор.

Завантаження Python

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

Середовище програмування для Python

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

Текстові редактори та інтегровані середовища програмування для Python:

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

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

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

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

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

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

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

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

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Хмарна телефонія затребувана в сегменті малого й середнього бізнесу, і ця тенденція спостерігається в останні два-три роки. Основними перевагами хмарних рішень були й залишаються швидке розгортання й мінімальні первісні інвестиції. У сформованій економічній ситуації дані переваги є основними драйверами росту попиту на програмні й хмарні рішення. Постійно зростаючий інтерес до них сприяв збільшенню наших продажів SIP-терміналів, що склав 15% стосовно минулого року.

Віртуальна АТМ не тільки дає можливість швидко й недорого побудувати корпоративну систему зв'язку, але й поєднує в собі переваги VoIP для бізнесу й моделі SaaS. А інтеграція бізнес-додатків з телефонією дозволяє задіяти в бізнес-процесах нові можливості CRM і інших інформаційних систем. Таким чином, АТМ – не просто хмарний аналог класичної телефонної системи: інтегровані з нею інструменти призначені для рішення різноманітних завдань, таких як аналітика телефонних дзвінків, статистика продажів і ін. Провайдери АТМ всі частіше роблять ставку на сервіси, що допомагають клієнтам заощаджувати й підвищувати ефективність бізнес-процесів.

По оцінках компанії Altegro, в останні три роки український ринок АТМ росте дуже швидко – на 15–25% щорічно. В J’son & Partners Consulting говорять про 30%, а деякі аналітики – і про 40%. Триваючий ріст хмарного сегмента на тлі падаючого IT-ринку – характерна тенденція останнього років.

За даними компанії Parallels, що опублікувала своє щорічне дослідження ринку хмарних послуг, популярних у малих і середніх українських компаній (основних користувачів послуг АТМ), за рік сукупний обсяг витрат на хмарні додатки й інфраструктуру виріс на 32% і склав 20,5 млрд грн. Найбільш

затребуваними виявилися IaaS (7,8 млрд грн.), SaaS (7,4 млрд грн.) і рішення для Web-присутності й Web-додатків (4,2 млрд грн.). Parallels прогнозує 40-процентне щорічне збільшення обсягу українського хмарного ринку на тлі зростаючого інтересу малих підприємств до хмарних технологій і підвищення загальної поінформованості про їхні можливості.

Попит на рішення по організації спільної роботи й комунікаціям помітно менший (1,1 млрд грн.), але вони мають найбільший потенціал росту. Як очікується, обсяг цього ринку буде збільшуватися на 63% у рік. Однак віртуальними АТМ користуються лише 3% підприємств SMB, близько 80% не використовують АТМ зовсім, в 17% немає навіть корпоративного телефону. При цьому 47% планують підключитися до віртуального АТМ у найближчі три роки.

Відзначимо найбільш важливі тенденції ринку АТМ:

– Попит на додаткові сервіси, інтеграцію із системами CRM, операторськими центрами, бізнес-додатками й хмарними платформами для спільної роботи. АТМ стає одним з компонентів комплексного рішення, називаного віртуальним або хмарним офісом.

– Інтеграція мобільного зв'язку й хмарної АТМ, єдине рішення для фіксованого й мобільного зв'язку (FMC). У деяких реалізаціях забезпечується повна інтеграція мобільних абонентів у корпоративну систему телефонії.

– Основним визначальним фактором при виборі провайдеру АТМ стає не ціна, а функціональність – наявність корисних для конкретного бізнесу сервісів, таких, наприклад, як аналітика даних, що накопичуються.

– Розширення кола замовників: поряд із представниками малого бізнесу до сервісів АТМ починають звертатися регіональні відділення й всі частіше – великі компанії. Серед користувачів хмарної телефонії збільшується й число компаній з регіонів, де росте рівень проникнення Інтернету.

Міняється й конкурентне середовище на українському ринку АТМ. Ці послуги тепер пропонують багато операторів зв'язку.

Відразу хочу сказати одну річ, SaaS – це спеціалізований термін і, якщо починати в ньому копатися, складний. Він може навести не тільки тугу, але й певний побожний жах. Але насправді штука це проста. Щоб користуватися якимось SaaS-продуктом, у тому числі нашим, розуміти глибинний, так навіть і поверхневий зміст даного терміна зовсім не обов'язково. Більше того, коли uCoz створювався, ніхто із самих розроблювачів навіть не чув подібного слова. Просто з'являлися продукти, що мають по своїй суті загальну модель, і потім її стали якось класифікувати, виділяти особливості, плюси, мінуси.

Якщо своїм покликанням або захопленням ви зробили веб-технології, або ж замислюєтеся про те, який тип продуктів особисто для вас підходить найбільше, які найбільш перспективні й т.п., все це може бути цікаво й корисно.

Про SaaS написано багато, і почати вивчення можна, наприклад, зі статті у Вікіпедії. Але, як правило, вся ця інформація звідти досить важко сприймається. У підсумку начебто б про що мова й зрозуміло, але навіщо треба людині залишається неясно. А якщо попросиш розповісти людини, про що він тільки що прочитав, то, знову ж, далеко не кожний зможе. Виходить класична ситуація: “Усе розумію, але пояснити не можу”.

Тому спробую спростити максимально й розглянути модель надання програмного забезпечення як сервісу, як послуги, на найпростіших прикладах.

Часто SaaS розглядається в якості бізнес-моделі, при цьому його найчастіше помилково дорівнюють до оренди, що хоч і носить загальні риси, але основна суть все-таки в іншому. При цьому, як правило, цікавим є технологічна особливість моделі, а не порядок і періодизація оплати.

Коли ми споживаємо ту або іншу послугу як сервіс, а не в якості встановлюваного в себе програмного забезпечення, ми вже в цей момент, як правило, стаємо споживачами SaaS. Найпростіший, і всіма використовуваний SaaS-сервіс – це сервіс електронної пошти – той же gmail.

Щоб організувати роботу електронної пошти самостійно з нуля, необхідно:

– спочатку налаштувати сервер;

– на ньому встановити спеціалізоване програмне забезпечення, якийсь агент передачі пошти (наприклад, Postfix), налаштувати все це, і в майбутньому обслуговувати;

– далі в себе на комп'ютері встановити й налаштувати поштовий клієнт, наприклад, The bat або Thunderbird;

– безумовно, за всім цим прийде стежити, що б продовжувало працювати, не заносилося в блек аркуші інших поштових серверів, боротися на своїй стороні зі спамом і т.д. і т.п.

Властиво, у багатьох організаціях пошта подібним чином сьогодні й працює, її обслуговує штатний сисадмин, або навіть цілий штат фахівців, залежно від розмірів організації і її інфраструктури.

Кінцеві користувачі й масовий споживач нічим таким не користується. Не користується він і поштою свого провайдеру, хоча ще років 12 назад майже всі використовували саме її, а пошту забирали винятково по pop3. Веб-інтерфейси для роботи з поштою були в зародковому стані. Сьогодні ж абсолютна більшість користується поштою через веб. І навіть люди, яких ніжно прийнято називати гиками, навіть вони пересаджуються найчастіше на веб-інтерфейс із поштових клієнтів. Це не виходить, що все в підсумку незабаром перейдуть до такої моделі споживання пошти, бувають специфічні завдання, буває дика сила звички, але для більшості вже сьогодні зручно й практичніше саме так. Тут усе: і надійність, і простота й навіть безпека грає на руку. Переходять на такі рішення не тільки приватні користувачі, але й цілі компанії, у деяких випадках навіть корпорації. А спеціалізовані рішення для корпоративної пошти на базі сервісу є як усе в того ж Google, так і в Zoho і ряду інших.

Пошта, по великому рахунку, це найпростіший приклад, і самий масовий. Хоча б у силу того, що абсолютна більшість користувачів мережі електронною поштою користуються, а от які-небудь CRM, ERP системи, або ті ж самі сайтбілдери потрібні далеко не всім. У них більше вузьке коло споживачів. Але

вже дуже багато програмних продуктів можна знайти у вигляді сервісів. У деяких випадках вони, звичайно, ще відстають від своїх десктопних аналогів, але найчастіше дають своєму споживачеві переваги. Майже завжди це виражається саме в простоті експлуатації, у відпаданні потреб по обслуговуванню, в економічній доцільності, попросту говорячи, у дешевині такого рішення й підходу. Інші плюси залежать в основному вужі від конкретної сфери, у деяких випадках це більше високий ступінь безпеки, в інших безпрецедентно зручна синхронізація й доступність комфортної багатокористувальницької роботи, за рахунок хмарного зберігання даних.

Що б не розпорошаться на загальні слова й характеристики, пропоную подивитися на прикладах. Самих таких повсякденних.

Частіше інших, як приклади використання SaaS рішень, можна зустріти системи керування проектами, і спільної роботи над ними, онлайнові органайзери, системи документообігу. Вони всі вже під рукою й багато хто ними вже користуються, не замислюючись над ідеологією таких сервісів і страшних розумних абревіатур.

Як я й говорив, за прикладами далеко ходити не треба. Робота з документами? Будь ласка – це популярний google docs, що дозволяє вам відмовитися від Word, Excel, і одержати ряд переваг, у першу чергу пов'язаних з можливостями спільної роботи над документами. Причому такі рішення є в цілого ряду компаній – є й в Microsoft, і в компанії Zoho і інших.

Онлайн органайзерів у принципі цілком достатньо, як і взагалі систем для організації роботи, ведення todo. Взяти хоча б наш календар і систему по керуванню проектам у вебтопі, або аналогічні рішення від google, або прославлені продукти компанії 37сигналів: basecamp, backpack.

Третій (не очевидний) приклад – онлайн ігри. Їх, звичайно, не прийнято відносити до SaaS-рішень, але й вони на сьогодні сталі доступні як сервіси, із усе тією же ідеологією. Найчастіше це MMORPG. Про всякий випадок нагадаю що

гри – це такі ж програми, а, приміром, Xbox Live Microsoft дуже чітко себе класифікує як SaaS-продукт.

До чого були всі ці приклади? До того, що програмне забезпечення, представлене як сервіс, давно навколо нас. І ринок, і ми самі використовуємо SaaS набагато частіше, ніж замислюємося. А зручність, простота, економічність та інші плюси SaaS завойовують серця споживачів, не пояснюючи їм свій складний пристрій і філософію моделі.

3.2 Розробка структурної схеми

Оператори активно просувають інноваційні бізнес-сервіси, включаючи й віртуальні АТМ. Так, існує проект віртуальної АТМ у всіх регіонах України. Вона дозволяє об'єднати SIM-карти й налаштувати безкоштовну переадресацію між ними. Всі дзвінки записуються й доступні для прослуховування в особистому кабінеті. Паралельно клієнти можуть підключити SIP-телефони або програмні телефони на комп'ютері. Як затверджує оператор, це перша в України повноцінна віртуальна АТМ, де мобільна й офісна телефонія об'єднані в одному рішенні.

Практично всі оператори зв'язку надають послуги АТМ із приблизно схожими функціями. При цьому сама віртуальна АТМ звичайно пропонується безкоштовно, а оператори традиційно заробляють на продажі номерів і телефонного трафіку. Для кінцевих замовників така ситуація має свої плюси й мінуси. З одного боку, замовникові не потрібно платити за функції віртуальної АТМ, тому що він неї одержує безкоштовно на додаток до телефонних номерів і трафіку. З іншого боку – ці функції прив'язані винятково до телефонних номерів і тарифів конкретного оператора. Бізнесу ж найчастіше потрібні більше широкі можливості комунікацій, підключення відразу декількох телефонних номерів від декількох операторів, інтеграція з іншими АТМ і бізнес-сервісами.

Для мобільних операторів надання нових послуг – це ключ до невичерпних джерел прибутку. Не секрет, що вони прагнуть вийти на нові ринки,

тому що більшість приватних клієнтів неохоче міняють тарифи. Ми активно співробітничаємо з операторами стільникового зв'язка як постачальники прикіцевого встаткування для їхніх послуг, а саме SIP-терміналів. Крапкою росту для операторів я б назвав продаж устаткування. Якщо ще недавно вони фокусувались тільки на самій послузі, то тепер зацікавлені в продажі комплексного рішення – «послуга + термінали». Причому встаткування не «дарується» покупцеві послуги, як це було раніше, а приносить операторам істотний дохід».

Тим часом на ринку АТМ в операторів з'являються конкуренти. Найбільш яскравий приклад – хмарна АТМ Інтернет-компанії. До складу Інтернет-компанії для малого бізнесу входять IVR, налаштування правил переадресації вхідних викликів, деталізація дзвінків, створення кнопки «Дзвінок із сайту», мобільна версія для iOS і Android. У платній версії доступна підтримка черги вхідних викликів, запис розмов і статистика звернень. За півроку роботи сервісу до послуги віртуальної АТМ удалося підключити більше 3 тис. замовників.

Цей сервіс уже перетерпів ряд змін. Зокрема, спрощена процедура реєстрації клієнтів, управляти викликами тепер можна в мобільному додатку (переклад, консультативний дзвінок, конференція), розширені можливості кастомізації. Надалі планується ще більше розширення не тільки функціональності, але й географії сервісу.

Класичні хмарні АТМ уже не задовольняють потреби сучасного бізнесу, оскільки «чиста» телефонія вже не є повноцінним засобом комунікацій. Наприклад, ще кілька років назад замість дзвінка по стаціонарному телефоні люди стали спілкуватися за допомогою мессенджерів. Skype використовується для відеоконференцій і вилучених співбесід, за допомогою Cisco WebEx проводяться вебінари й дистанційне навчання, а співробітники, що перебувають поза офісом, дзвонять за допомогою мобільного телефону.

Ні одну із цих завдань класична хмарна АТМ не вирішує, внаслідок чого компаніям доводиться використовувати великий набір різнорідних сервісів, на які

корпоративна політика не поширюється, у тому числі відносно безпеки. Проблема дозволяється шляхом переходу до так званих уніфікованих комунікацій, тепер доступним і із хмари.

Системи уніфікованих комунікацій (UC) поєднують в одному рішенні основні засоби зв'язку. Типовим представником є Microsoft Skype for Business, що теж доступний із хмари. Користувачі одержують функції єдиної адресної книги зі статусами присутності й фотографіями абонентів, миттєві повідомлення й чати, аудіо- і відеодзвінки, аудіо- і відеоконференції, Web-конференції, спільний доступ до робочого стола й додатків, а також повний набір можливостей хмарної IP-АТМ, включаючи групи викликів, визначення номерів, функції автосекретаря, голосової пошти, переадресації виклику й т.д.

Сервіс доступний з будь-якого пристрою (комп'ютера, смартфона, планшета або IP-телефону) і з будь-якого місця, де є вихід в Інтернет. Співробітники можуть дзвонити з будинку, з іншого міста або країни. Наймати на роботу надомних співробітників, а також відкривати філії можна без додаткових витрат на телефонію. А сам сервіс Skype інтегрується з такими корпоративними сервісами, як Active Directory і Exchange. Замовниками затребувані також можливість підключення власних телефонних номерів і інтеграція із уже наявними в компанії АТМ, що не підтримується багатьма хмарними АТМ.

Подальший розвиток комунікаційних сервісів іде в напрямку ще більшої уніфікації й інтеграції з іншими корпоративними сервісами-бізнесами-сервісами. Так, наприклад, в 2017 році з'явилася можливість інтеграції хмарних АТМ із Office 365. Серед самих затребуваних функцій віртуальних АТМ я б назвав запис мови й голосові меню, анонс часу очікування, а також інтеграцію з CRM-системами й CTI. Все це із просунутої й дорогої функціональності операторського центра, доступної колись тільки великим корпоративним замовникам, перетворюється в рядовий запит ринку, у тому числі з боку сегмента SMB. Навіть самі бюджетні моделі IP-АТМ Panasonic надають такі сервіси. І хмарні сервіси не виключення. Раніше бізнес-платформа будувалася навколо телефонії. З розвитком сучасних технологій, зокрема із широким поширенням IP,

змінився підхід до телефонії. Основною платформою стає ПЗ, що обслуговує бізнес. А телефонія тепер є лише одним з видів комунікацій, інтегрованим у бізнес-додаток».

АТМ популярні в сегменті малого й середнього бізнесу. Великий бізнес як і раніше насторожено ставиться до хмарних сервісів, та й послуги аутсорсингу в цілому в України сприймають із недовірою. Це більшою мірою пов'язане з вимогами безпеки. Однак варто відзначити, що саме хмарні сервіси в найближчому майбутньому ще більше потіснять традиційні рішення.

Основною цільовою аудиторією як хмарних сервісів у цілому, так і хмарних АТМ зокрема є малі й середні компанії, особлива увага звертається зниженню витрат. Великий бізнес, як правило, використовує власну інфраструктуру, у яку вже інвестовані значні засоби. Крім того, великі підприємства найчастіше висувають досить серйозні вимоги по кастомізації сервісів, що не завжди можливо для стандартизованого хмарного сервісу.

Проте ми бачимо ріст попиту на хмарні АТМ і з боку великих організацій, і тому є ряд причин:

– По-перше, рано або пізно власна телефонна інфраструктура починає застарівати, а її функціональні можливості перестають задовольняти зростаючі потреби.

– По-друге, надійність застаріваючої інфраструктури знижується, а ремонт і підтримка стають усе дорожче.

– По-третє, компанія може зштовхнутися з обмеженнями ємності АТМ. Отоді й виникає питання про глобальну заміну всієї телефонної інфраструктури».

Оскільки великий бізнес – це, як правило, більші територіально розподілені компанії, де замінити треба не одну, а кілька десятків телефонних станцій, хмарні АТМ виявляються вигідним рішенням. Якщо враховувати сукупну вартість володіння, ставку дисконтування, а також економію, пов'язану з масштабуванням сервісу і його підтримкою, то хмарні АТМ можуть виявитися дешевше навіть у довгостроковій перспективі (п'ять і більше років).

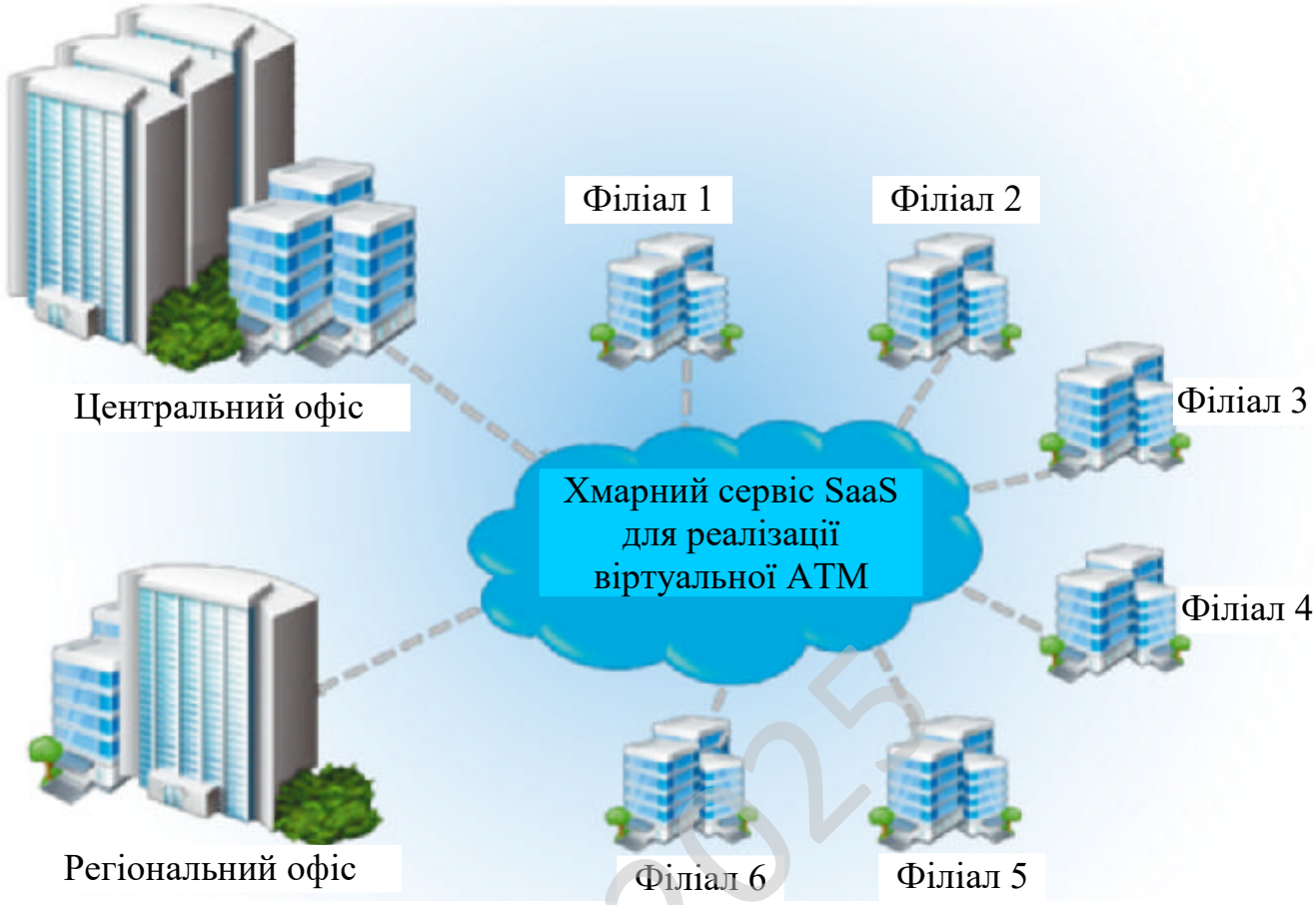

Рисунок 3.1 – Структурна схема системи

Розглянемо основні алгоритми кодування мови, які використовуються в забезпечення хмарного сервісу SaaS для реалізації віртуальної АТМ.

Кодери вихідної інформації (вокодери) і гібридні алгоритми

Багато методів кодування використовують особливості людської мови, зв'язані з будовою голосового апарата. Кодери, у яких реалізуються такі методи, називають кодерами вихідної інформації або вокодерами (voice coding).

Послідовність формованих звуків становить тонову мову. Якщо звук формується байдужості зв'язувань, тон у ньому відсутній, і послідовність таких звуків становить нетонову мову. Спектр тонового звуку може бути змодельований шляхом подачі спеціальним образом сформованого сигналу порушення на вхід цифрового фільтра з параметрами, обумовленими декількома дійсними коефіцієнтами. Спектр нетонових звуків – практично рівномірний, що

обумовлено їхнім шумовим характером. У реальних мовних сигналах не всі звуки можна чітко розділити на тонові і нетонові, а доводиться мати справу з якимись перехідними варіантами, що утрудняє створення алгоритмів кодування, які забезпечують високу якість передачі мови при низької швидкості передачі інформації. Описаний принцип кодування одержав назву LPC (Linear Prediction Coding – кодування з лінійним пророкуванням), оскільки центральним елементом моделі голосового тракту є лінійний фільтр. Алгоритми кодування форми сигналу засновані на наявності кореляційних зв'язків між відліками сигналу, які дають можливість лінійного пророкування. У сполученні з адаптивним квантуванням цей підхід дозволяє забезпечити гарну якість мови при швидкості передачі біт порядку 24-32 Кбіт/с. LPC-кодери (вокодери) використовують просту математичну модель голосового тракту й дозволяють використовувати дуже низькі швидкості передачі інформації 1200-2400 біт/с, однак ціною «синтетичного» характеру мови.

Гібридні алгоритми кодування й алгоритми типу «аналіз шляхом синтезу» (ABS) являють собою спроби сполучити позитивні властивості двох описаних вище основних підходів і будувати ефективні схеми кодування з діапазоном швидкостей передачі біт 6-16Кбіт/с. Важлива відмінність кодерів такого типу полягає в тому, що в рамках цих алгоритмів немає необхідності ухвалювати рішення щодо типу відтвореного звуку (тонов або нетоновий), тому що передбачаються спеціальні міри для кодування сигналу помилки після проходження порушення через LPC-фільтр [11-15].

Процесори цифрової обробки сигналів для мовних кодексів

Процесори DSP (цифрової обробки сигналів) мають архітектуру, оптимізовану для виконання операцій, які характерні для типових алгоритмів обробки сигналів. Архітектура процесорів DSP часто характеризується наявністю декількох обчислювальних блоків, що забезпечують виконання одночасних операцій в одному такті роботи процесора. Для завантаження обчислювальних блоків даними передбачається кілька шин передачі даних і багатопортова пам'ять

даних. Для збільшення продуктивності пам'ять інструкцій і пам'ять даних розділені, а доступ до них здійснюється також по роздільних шинах. Для процесорів DSP характерне використання інструкцій збільшеної довжини, що містять поля для керування всіма обчислювальними блоками [11-15].

Алгоритми кодування

У першу чергу необхідно зрозуміти, якими критеріями потрібно керуватися при виборі кодека для використання в IP-телефонії [15-22].

Швидкість передачі, що передбачають наявні сьогодні вузькополосні кодеки, лежить у межах 1.2 – 64 Кбіт/с. Природно, що від цього параметра прямо залежить якість відтвореної мови. У рамках існуючих технологій якість ТфОП (toll quality) неможливо забезпечити при швидкостях менш 5 Кбіт/с.

Придушення періодів мовчання (VAD, CNG, DTX). Детектор мовної активності (VAD) необхідний для визначення періодів часу, коли користувач говорить. Підтримка переривчастої передачі (DTX) дозволяє кодеку припинити передачу пакетів у той момент, коли VAD виявив період мовчання. Генератор комфортного шуму (CNG) служить для генерації фонового шуму.

Більшість вузькополосних кодеків обробляють мовну інформацію блоками, називаними кадрами (frames), і тому необхідно здійснювати попередній аналіз обчислень, що випливають безпосередньо за обчисленнями в блоці, що вони в цей момент кодують. Розмір кадру важливий, тому що мінімальна теоретично досяжна затримка передачі інформації (алгоритмічна затримка) визначається сумою цього параметра й довжини буфера попереднього аналізу. У дійсності процесори цифрової обробки сигналів, які виконують алгоритм кодування, мають кінцеву продуктивність, так що реальна затримка сигналу більше теоретичної. До кадру, сформованому кодеком, додається безліч додаткової інформації – заголовки IP (20 байтів), UDP (8 байтів), RTP (12 байтів). Для кодеку із тривалістю кадру 30 мс посилка таких кадрів по мережі привела б до передачі надлишкової інформації зі швидкістю 10.6 кбіт/с, що перевищує швидкість передачі мовної інформації в більшості вузькополосних кодеків.

Тому звичайно використовується пересилання декількох кадрів у пакеті, при цьому їхня кількість обмежена максимально припустимою затримкою. У більшості випадків в одному пакеті передається до 60 мс мовної інформації. Чим менше тривалість кадру, тим більше кадрів доводиться впаковувати в один пакет, тобто затримка визначається зовсім не довжиною кадру, а практично прийнятним обсягом корисного навантаження в пакеті. Крім того, кодеки з більшою довжиною кадру більше ефективні, тому що тут діє загальний принцип: чим довше спостерігається явище (мовний сигнал), тим краще воно може бути змодельоване.

Втрати пакетів є невід'ємним атрибутом IP-мереж. В зв’язку з тим, що пакети містять кадри, сформовані кодеком, то це викликає втрати кадрів. Але втрати пакетів і втрати кадрів не обов'язково прямо зв'язані між собою, тому що існують підходи, що дозволяють зменшити число загублених кадрів при даному числі загублених пакетів. Додаткова службова інформація розподіляється між декількома пакетами, так що при втраті деякого числа пакетів кадри можуть бути відновлені. Однак позитивний ефект від введення надмірності для боротьби із втратами пакетів не настільки легко досяжний, оскільки втрати в IP-мережах відбуваються пачками, тобто значно більш імовірно те, що буде загублено відразу кілька пакетів підряд, ніж те, що загублені пакети розподіляться в послідовності переданих пакетів по одному. Тому якщо застосовувати прості схеми введення надмірності, то в реальних умовах вони, хоча й збільшать обсяг надлишкової інформації, але, швидше за все, виявляться марними. Крім того, введення надмірності негативно позначається на затримці відтворення сигналу.

3.3 Розробка функціональної схеми

На функціональній схемі, зображеній на рисунку 3.2, є можливість прослідкувати за можливими шляхами проходження сигналів від одного функціонального блоку до іншого. Тонка стрілочка вказує шлях проходження сигналів між функціональними блоками, товста – мережна взаємодія.

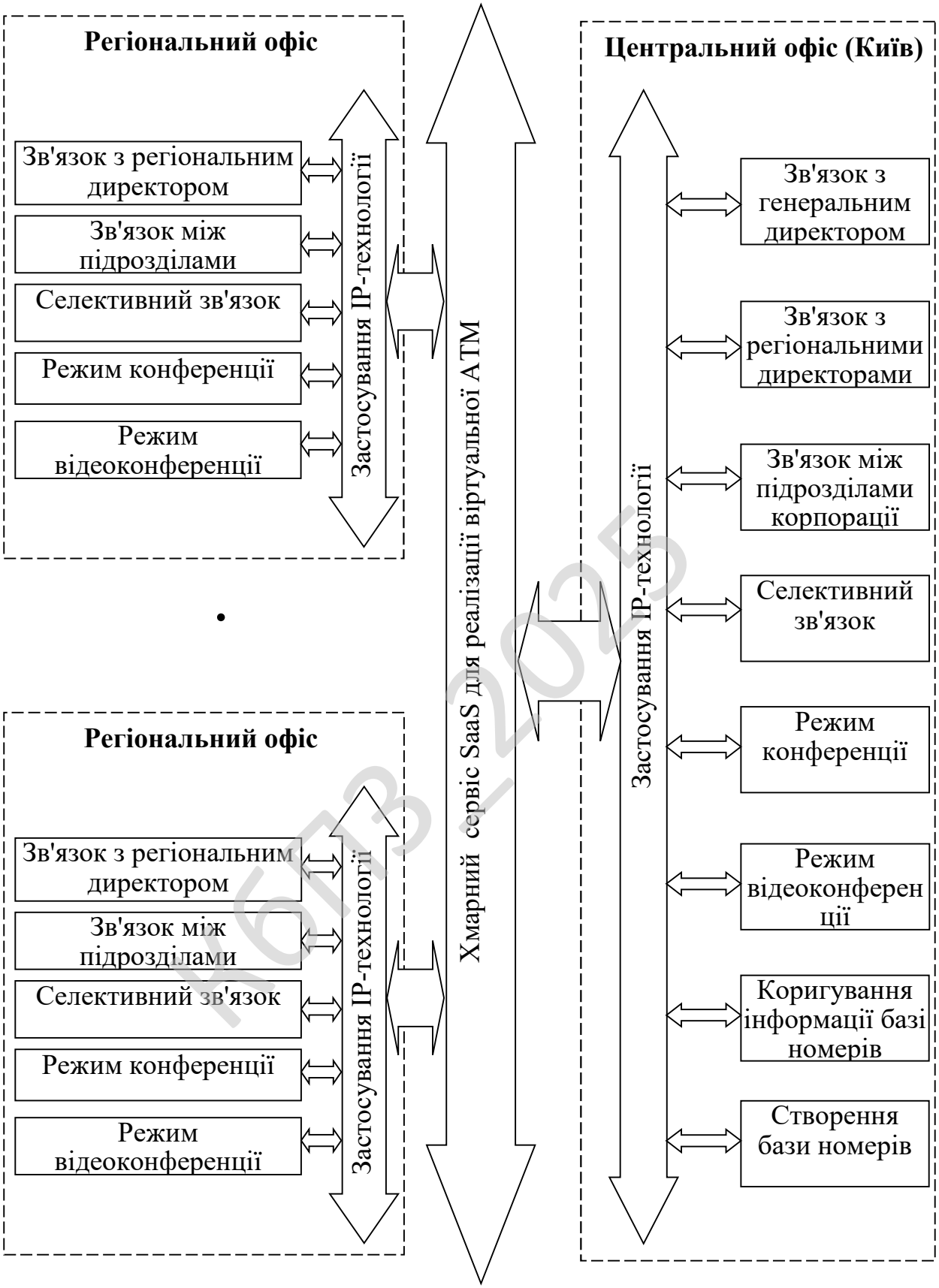

Рисунок 3.2 – Функціональна схема системи

Розглянемо основні функціональні взаємодії між центральним офісом та регіональними офісами. З центрального офісу з застосуванням IP технології проводиться зв'язок з регіональними офісами, такими абонентами як – регіональний директор, підрозділи. Є можливість проведення різних режимів зв`язку при різній кількості абонентів з обліком абонентів (за допомогою БД).

Архітектура технології VoIP може бути спрощено представлена у вигляді двох площин. Нижня площина – це базова мережа з маршрутизацією пакетів IP, верхня площина – це відкрита архітектура керування обслуговуванням викликів.

Нижня площина являє собою комбінацію відомих протоколів Інтернет: це – RTP, що функціонує поверх протоколу UDP, розташованого, у свою чергу, у стеці протоколів TCP/IP над протоколом IP. Таким чином, ієрархія RTP/UDP/IP являє собою свого роду транспортний механізм для мовного трафіку. У мережах з маршрутизацією пакетів IP для передачі даних завжди передбачаються механізми повторної передачі пакетів у випадку їхньої втрати. Рекомендації ITU-T допускають затримки в одному напрямку не перевищуючі 150 мс.

Розглянемо протокол TCP/IP. Transmission Control Protocol/Internet Protocol – це промисловий стандарт стеку протоколів, розроблений для глобальних мереж:

– це найбільш завершений стандартний і водночас популярний стек мережних протоколів, що має багаторічну історію.

– майже усі великі мережі передають основну частину свого трафіка за допомогою протоколу TCP/IP.

– це метод одержання доступу до мережі Internet.

– стек TCP/IP є основою для створення intranet-корпоративної мережі, що використовує транспортні послуги Internet і гіпертекстову технологію WWW, розроблену в Internet.

– усі сучасні операційні системи підтримують стек TCP/IP.

– це гнучка технологія для з'єднання різнорідних систем як на рівні транспортних підсистем, так і на рівні прикладних сервісів.

– це масштабована міжплатформенна середа для додатків клієнт-сервер.

Протоколи TCP/IP поділяються на 4 рівні.

Найнижчий IV рівень відповідає фізичному і канальному рівням моделі OSI. Цей рівень у протоколах TCP/IP підтримує всі популярні стандарти фізичного і канального рівня: для локальних мереж це Ethernet, Token Ring, FDDI, Fast Ethernet, 100VG-AnyLAN, для глобальних мереж – протоколи з'єднань "точка-точка" SLIP і PPP, протоколи територіальних мереж із комутацією пакетів X. 25, frame relay. Звичайно тільки з'являється нова технологія локальних або глобальних мереж вона швидко включається в стек TCP/IP за рахунок розробки відповідного стандарту, що визначає метод інкапсуляції пакетів IP у її кадри.

Наступний III рівень – це рівень міжмережної взаємодії, що займається передачею пакетів із використанням різноманітних транспортних технологій локальних мереж, територіальних мереж, ліній спеціального зв'язку тощо. У якості основного протоколу мережного рівня використовується протокол IP.

Наступний II рівень називається основним. На цьому рівні функціонує протокол керування передачею TCP і протокол дейтаграм користувача UDP. Протокол TCP забезпечує надійну передачу повідомлень між віддаленими прикладними процесами за рахунок утворення віртуальних з'єднань. Протокол UDP забезпечує передачу прикладних пакетів дейтаграмним засобом, як і IP, і виконує тільки функції сполучного ланка між мережним протоколом і численними прикладними процесами.

Верхній I рівень називається прикладним. За довгі роки використання в мережах різноманітних країн і організацій стек TCP/IP накопичив велику кількість протоколів і сервісів прикладного рівня. До них відносять такі широко використовувані протоколи, як протокол копіювання файлів FTP, протокол емуляції термінала telnet, поштовий протокол SMTP, використовуваний в електронній пошті мережі Internet, гіпертекстові сервіси доступу до віддаленої інформації, такі як WWW і багато інших.

Основу транспортних засобів стека протоколів TCP/IP складає протокол міжмережної взаємодії – Internet Protocol (IP). Основні функцій протоколу IP:

– перенос між мережами різноманітних типів адресної інформації в уніфікованій формі;

– складання і розбирання пакетів при передачі їх між мережами з різноманітним максимальним значенням довжини пакета.

Пакет IP складається з заголовка і поля даних. Максимальна довжина поля даних пакета обмежена розрядністю поля, що визначає цей розмір, і складає 65535 байтів, проте при передачі по мережах різноманітного типу довжина пакета вибирається з урахуванням максимальної довжини пакета протоколу нижнього рівня, що несе IP-пакети. Якщо це кадри Ethernet, то вибираються пакети з максимальною довжиною в 1500 байтів, що поміщаються в поле даних кадру.

Задачею протоколу транспортного рівня UDP є передача даних між прикладними процесами без гарантій доставки, тому його пакети можуть бути загублені, продубльовані або прийти не в тому порядку, у якому відправлені.

У стеку протоколів TCP/IP протокол TCP працює так само, як і протокол UDP, на транспортному рівні. Він забезпечує надійне транспортування даних між прикладними процесами шляхом установлення логічного з'єднання.

У протоколі TCP для зв'язку з прикладними процесами використовуються порти. Номера портам присвоюються. Є стандартні, зарезервовані номера (наприклад, номер 21 закріплений за сервісом FTP, 23 – за telnet), а менше відомі додатки користуються довільно обраними локальними номерами.

Перейдемо до верхньої площини керування обслуговуванням запитів зв'язку. Керування обслуговуванням виклику передбачає прийняття рішень про те, куди виклик повинен бути спрямований, і яким образом повинне бути встановлене з'єднання між абонентами. Інструмент такого керування – телефонні системи сигналізації, починаючи із систем, підтримуваних декадно-кроковими АТС і функцій, що передбачають об'єднання, маршрутизації й функцій створення розмовного каналу, що комутирується, у тих самих декадно-крокових шукачах.

Далі принципи сигналізації еволюціонували до систем сигналізації по виділених сигнальних каналах, до багаточастотної сигналізації, до протоколів загальканальної сигналізації [6, 7] і до передачі функцій маршрутизації у відповідні вузли обробки послуг мережі [8].

У мережах з комутацією пакетів ситуація більше складна. Мережа з маршрутизацією пакетів IP принципово підтримує одночасно цілий ряд різноманітних протоколів маршрутизації. Такими протоколами на сьогодні є: RIP, IGRP, EIGRP, IS-IS, OSPF, BGP і ін. Точно так само й для IP-телефонії розроблений цілий ряд протоколів

Найпоширенішим є протокол H.323, зокрема, тому, що він став застосовуватися раніше інших протоколів. Інший протокол площини керування обслуговуванням виклику -SIP – орієнтований на те, щоб зробити кінцеві пристрої й шлюзи більш інтелектуальними й підтримувати додаткові послуги для користувачів. Ще один протокол – SGCP – розроблявся, для того, щоб зменшити вартість шлюзів за рахунок реалізації функцій інтелектуальної обробки виклику в централізованому встаткуванні. Протокол IPDC дуже схожий на SGCP, але має більше, ніж SGCP, механізмів експлуатаційного керування (OAM&P). Існує більш функціональний, ніж MGCP, протокол MEGACO. Його адаптований до H.323 варіант в рекомендації H.248.

Щоб стало зрозуміло, чим конкретно відрізняються один від одного перераховані в попередньому параграфі протоколи, розглянемо архітектуру мереж, побудованих на базі цих протоколів, і процедури встановлення й завершення з'єднання з їхнім використанням.

Мережі на базі протоколів H.323 орієнтовані на інтеграцію з телефонними мережами й можуть розглядатися як мережі ISDN, накладені на мережі передачі даних. Рекомендація H.323 передбачає досить складний набір протоколів, що призначений не просто для передачі мовної інформації з IP-мереж з комутацією пакетів. Його мета – забезпечити роботу мультимедійних додатків у мережах з негарантованою якістю обслуговування. Мовний трафік – це тільки один з

додатків H.323, поряд з відеоінформацією й даними. Варіант побудови мереж IP-телефонії в рекомендації H.323, добре підходить тим операторам місцевих телефонних мереж, які зацікавлені у використанні мережі з комутацією пакетів (IP-мережі) для надання послуг міжміського й міжнародного зв'язку.

Основними пристроями мережі є: термінал (Terminal), шлюз (Gateway), воротар (Gatekeeper) і пристрій керування конференціями (MCU).

У сценарії встановлення з'єднання між двома користувачами передбачається, що кінцеві користувачі вже знають IP-адреси один одного. У звичайному випадку етапів буває більше, оскільки у встановленні з'єднання беруть участь gatekeeper і й шлюзи.

Розглянемо крок за кроком цей спрощений сценарій.

1. Кінцевий пристрій користувача А надсилає запит з'єднання – повідомлення SETUP – до кінцевого пристрою користувача В.

2. Кінцевий пристрій викликуваного користувача В відповідає на повідомлення SETUP повідомленням ALERTING, що означає, що пристрій вільний, а викликуваному користувачеві подається сигнал про вхідний виклик.

3. Після того, як користувач У приймає виклик, до зухвалої сторони А передається повідомлення CONNECT з номером TCP-порту каналу H.245.

4. Кінцеві пристрої обмінюються по каналі H.245 інформацією про типи використовуваних мовних кодеків та про інші функціональні можливості встаткування, і сповіщають один одного про номери портів RTP, на які варто передавати інформацію.

5. Відкриваються логічні канали для передачі мовної інформації.

6. Мовна інформація передається в обидва боки в повідомленнях протоколу RTP; крім того, ведеться контроль передачі інформації за допомогою RTCP.

Наведена процедура обслуговування виклику базується на протоколі H.323 версії 1. Версія 2 протоколи H.323 дозволяє передавати інформацію, необхідну для створення логічних каналів, безпосередньо в повідомленні SETUP

протоколу H.225.0 без використання протоколу H.245. Така процедура називається «швидкий старт» (Fast Start) і дозволяє скоротити кількість циклів обміну інформацією при встановленні з'єднання. Крім організації базового з'єднання, у мережах H.323 передбачене надання додаткових послуг відповідно до рекомендацій ITU H.450.X. Моніторинг якості обслуговування забезпечується протоколом RTCP, однак обмін інформацією RTCP відбувається тільки між кінцевими пристроями, що беруть участь у з'єднанні.

Другий підхід до побудови мереж IP-телефонії заснований на використанні протоколу SIP – Session Initiation Protocol. SIP являє собою текст – орієнтований протокол, що є частиною глобальної архітектури мультимедіа. Ця архітектура також містить у собі протокол резервування ресурсів (RSVP, RFC 2205), транспортний протокол реального часу (RTP, RFC 1889), протокол передачі потоків у реальному часі (RTSP, RFC 2326), протокол опису параметрів зв'язку (SDP, RFC 2327), протокол повідомлення про зв'язок (SAP). Однак функції протоколу SIP не залежать від кожного із цих протоколів.

Третій підхід до побудови мереж IP-телефонії, заснована на використанні протоколу MGCP. При розробці цього протоколу робоча група MEGACO опиралася на мережну архітектуру, що містить основні функціональні блоки трьох видів:

– шлюз – Media Gateway (MG), що виконує функції перетворення мовної інформації, що надходить із боку ТфОП з постійною швидкістю передачі, у вид, придатний для передачі по мережах з маршрутизацією пакетів IP (кодування й упакування мовної інформації в пакети RTP/UDP/IP, та зворотнє перетворення);

– контролер шлюзів – Call Agent, що виконує функції керування шлюзами;

– шлюз сигналізації – SIGNALING Gateway (SG), що забезпечує доставку сигнальної інформації, що надходить із боку ТфОП, до контролера шлюзів і перенос сигнальної інформації у зворотному напрямку.

Таким чином, весь інтелект функціонально розподіленого шлюзу зосереджений у контролері, функції якого можуть бути розподілені між декількома комп'ютерними платформами.

Для побудови добре функціонуючих і сумісних із ТфОП мереж IP-телефонії підходять протоколи H.323 і MGCP. Як вже відзначалось, протокол SIP трохи гірше взаємодіє із системами сигналізації, використовуваними в ТфОП. Підхід, заснований на використанні протоколу MGCP, має досить важливу перевагу перед підходом H.323: підтримка контролером шлюзів сигналізації ОКС7 і інших видів сигналізації, а також прозора трансляція сигнальної інформації з мережі IP-телефонії. У мережі, побудованої на базі рекомендації H.323, сигналізація ОКС7, як і будь-яка інша сигналізація, конвертується шлюзом у сигнальні повідомлення H.225.0 (Q.931). Основним недоліком третього з наведених у даному параграфі підходів є незакінченість стандартів. До недоліків можна віднести також відсутність стандартизованого протоколу взаємодії між контролерами. Крім того, протокол MGCP є протоколом керування шлюзами, але не призначений для керування з'єднаннями за участю термінального встаткування користувачів (IP-телефонів). Це означає, що в мережі, побудованої на базі протоколу MGCP, для керування термінальним устаткуванням повинен бути присутнім gatekeeper або сервер SIP. Варто також відзначити, що в існуючих додатках IP-телефонії, таких як надання послуг міжнародного й міжміського зв'язку, використовувати протокол MGCP (також, як і протокол SIP) недоцільно у зв'язку з тим, що основна кількість мереж IP-телефонії сьогодні побудована на базі протоколу H.323. В останньому зі згаданих підходів (у проекті версії 4 рекомендації H.323) введено принцип декомпозиції шлюзів, використаний у третьому підході. Керування функціональними блоками розподіленого шлюзу буде здійснюватися контролером шлюзу – MGC (Media Gateway Controller) за допомогою протоколу MEGACO/H.248.

Як було обґрунтовано раніше, сучасна IP-телефонія будується на основі протоколу H.323. Розглянемо більш докладно цей протокол. У рекомендаціях, що

входять у сімейство H.323, визначені протоколи, методи й мережні елементи, необхідні для організації мультимедійного зв'язку між двома або більше користувачами [15-22].

Найбільш затребуваною з послуг, специфікованих у рекомендації H.323, є послуга передачі мовної інформації з мереж з маршрутизацією пакетів IP. Найпоширенішим підходом до побудови мереж IP-телефонії сьогодні є саме підхід, запропонований ITU-T у рекомендації H.323.

Сімейство протоколів H.323 містить у собі три основних протоколи: протокол взаємодії кінцевого устаткування з gatekeeper – RAS, протокол керування з'єднаннями – H.225 і протокол керування логічними каналами – H.245. Ці три протоколи, разом з Інтернет-протоколами TCP/IP, UDP, RTP і RTCP, а також з описаним в [6] протоколом Q.931 складають основу технології IP-телефонії. Суть ієрархії цих протоколів полягає в наступному. Для переносу сигнальних повідомлень H.225 і керуючих повідомлень H.245 використовується протокол з встановленням з'єднання й з гарантованою доставкою інформації – TCP. Сигнальні повідомлення RAS переносяться протоколом з негарантованою доставкою інформації – UDP. Протокол RAS забезпечує контроль використання мережних ресурсів, підтримує автентифікацію користувачів і може забезпечувати нарахування плати за послуги. Для переносу мовної і відеоінформації використовується протокол передачі інформації в реальному часі – RTP. Контроль переносу користувальницької інформації виробляється протоколом RTCP. Процедура встановлення з'єднання в таких мережах IP-телефонії базується на рекомендації Q.931 [15] і аналогічна процедурі, використовуваної в ISDN.

Основними пристроями мережі є: термінал, шлюз, gatekeeper і пристрій керування конференціями.

Термінал H.323 – це кінцевий пристрій мережі IP-телефонії, що забезпечує двосторонній мовний або мультимедійний зв'язок з іншим терміналом, шлюзом або пристроєм керування конференціями. Користувальницький інтерфейс керування системою дає користувачеві можливість створювати й приймати

виклики, а також конфігурувати систему й контролювати її роботу.

Модуль керування підтримує три види сигналізації: H.225, H.245 і RAS. Цей модуль забезпечує реєстрацію термінала у gatekeeper, установлення й завершення з'єднання, обмін інформацією, необхідної для відкриття мовних каналів, надання додаткових послуг і техобслуговування.

Телематичні додатки забезпечують передачу користувальницьких даних, нерухливих зображень і файлів, доступ до баз даних і т.п. Стандартним протоколом для підтримки таких додатків є протокол T. 120.

Модуль H.225.0 відповідає за перетворення відеоінформації, мови, даних і сигнальної інформації у вид, придатний для передачі по мережах з маршрутизацією пакетів IP, і за зворотне перетворення. Крім того, функціями модуля є розбивка інформації на логічні кадри, нумерація послідовно переданих кадрів, виявлення й корекція помилок.

Мережний інтерфейс забезпечує гарантовану передачу керуючих повідомлень H.245, сигнальних повідомлень H.225.0 (Q.931) і користувальницьких даних за допомогою протоколу TCP і негарантовану передачу мовної й відеоінформації, а також повідомлень RAS, за допомогою протоколу UDP.

Блок синхронізації вносить затримку на прийомній стороні з метою забезпечити синхронізацію джерела інформації з її приймачем, узгодження мовних і відеоканалів або згладжування варіації затримки інформації.

Відеокодеки кодують відеоінформацію, що надходить від зовнішнього джерела відеосигналів (відеокамери або відеомагнітофона), для її передачі по мережі з маршрутизацією пакетів IP і декодують сигнали, що надходять із мережі, для наступного відображення відеоінформації на моніторі або телевізорі.

Аудіокодеки кодують аудіоінформацію, що надходить від мікрофона (або інших джерел аудіоінформації), для її передачі по мережі з маршрутизацією пакетів IP і декодують сигнали, що надходять із мережі, для відтворення.

Слід зазначити, що при організації децентралізованої конференції

термінал H.323 може приймати більш ніж один потік мовної інформації. У цьому випадку термінал повинен уміти змішувати або перемикати пакетовану мову, що надходить від інших учасників конференції.

Основною функцією шлюзу H.323 є перетворення мовної (мультимедійної) інформації, що надходить із боку ТфОП з постійною швидкістю, у вид, придатний для передачі по IP-мережах.

При відсутності в мережі gatekeeper повинна бути реалізована ще одна функція шлюзу – перетворення номера ТфОП у транспортну адресу IP-Мережі.

З боку мереж з маршрутизацією пакетів IP, так само, як і з боку ТфОП, шлюз може брати участь у з'єднаннях як термінал або пристрій керування конференціями. У випадку, коли термінал H.323 зв'язується з іншим терміналом H.323, розташованим у ті ж самій IP-мережі, шлюз у цьому з'єднанні не бере участь.

Шлюз, у сукупності з gatekeeper мережі IP-телефонії, утворить універсальну платформу для надання всього спектра послуг зв'язку.

У gatekeeper зосереджений весь інтелект мереж IP-телефонії, що базуються на рекомендації ITU H.323. Мережа H.323 має зонну архітектуру. Gatekeeper виконує функції керування зоною мережі IP-телефонії, у яку входять термінали, шлюзи й пристрої керування конференціями, зареєстровані в цього gatekeeper. Різні ділянки зони мережі H.323 можуть бути територіально рознесені й з'єднуватися один з одним через маршрутизатори. Варто звернути увагу на те, що комутатори кадрів Ethernet і маршрутизатори пакетів IP не є мережними елементами H.323, тому що вони працюють на ланковому або мережному рівнях відповідно, у той час як устаткування H.323 працює на прикладному рівні стека протоколів TCP/IP.

У число найбільш важливих функцій, виконуваних gatekeeper, входять:

– перетворення так званої alias-адреси (ім'я абонента, телефонного номера, адреси електронної пошти й ін.) у транспортну адресу мережі з маршрутизацією пакетів IP (IP адреса й номер порту TCP);

– контроль доступу користувачів системи до послуг IP-телефонії за допомогою сигналізації RAS (використовуються повідомлення ARQ/ACF/ARJ);

– контроль, керування й резервування пропускної здатності мережі;

– маршрутизація сигнальних повідомлень між терміналами, розташованими в одній зоні; gatekeeper може організовувати сигнальний канал безпосередньо між терміналами або ретранслювати сигнальні повідомлення від одного термінала до іншого.

У тому випадку, коли абонент, який викликає, знає IP-адресу термінала абонента, який викликається, з'єднання між двома пристроями може бути встановлене без допомоги gatekeeper. Але, при наявності gatekeeper я забезпечується мобільність абонентів, тобто здатність користувача одержати доступ до послуг з будь-якого термінала в будь-якому місці мережі й здатність мережі ідентифікувати користувачів при їхньому переміщенні з одного місця в інше. При відсутності в мережі gatekeeper перетворення адреси викликуваного абонента, що надходить із боку ТфОП у форматі E. 164, у транспортну адресу IP-мережі повинне виконуватися шлюзом. В одній мережі може перебувати декілька gatekeeper, які повинні взаємодіяти між собою. Слід особливо зазначити, що хоча gatekeeper є окремим логічним елементом мережі, він може бути реалізований у терміналі, у шлюзі, у пристрої керування конференціями або в пристроях, не специфікованих у рекомендації H.323.

Рекомендація H.323 передбачає три види конференцій.

Перший вид – централізована конференція, у якій кінцеві пристрої з'єднуються в режимі точка-точка із пристроєм керування конференціями (MCU), що контролює процес створення й завершення конференції, а також обробку потоків користувацької інформації.

Другий вид – децентралізована конференція, у якій кожний її учасник з'єднується з іншими учасниками в режимі точка – група точок, і кінцеві пристрої самі обробляють потоки інформації, що надходять від інших учасників.

Третій вид – змішана конференція, тобто комбінація двох попередніх

видів.

Перевага централізованої конференції – порівняно прості вимоги до термінального встаткування, недолік – більша вартість пристрою керування конференціями.

Для децентралізованої конференції потрібно більше складне термінальне встаткування, крім того, бажано, щоб у мережі підтримувалася передача пакетів IP у режимі багатоадресного розсилання (IP multicasting). Якщо мережа не підтримує цей режим, термінал може передавати інформацію до кожного з інших терміналів, що беруть участь у конференції, у режимі точка-точка, але це стає неефективним при числі учасників більше чотирьох. Пристрій керування конференціями MCU містить один обов'язковий елемент – контролер багатоточкових з'єднань – Multipoint controller (MC). Крім того, MCU може містити один або більше процесорів для обробки інформації користувачів при багатоточкових з'єднаннях – Multipoint processor (MP). Слід зазначити, що контролер MC і процесор MP є самостійними логічними пристроями H.323 і що контролер може існувати незалежно від процесора (зворотне невірно). Контролер може бути фізично сполучений з gatekeeper, зі шлюзом або з MCU, а MCU, у свою чергу, може бути сполучене зі шлюзом або з gatekeeper.

Контролер конференцій повинен використовуватися для організації конференції будь-якого виду. Він організує обмін між учасниками конференції даними про функціональні можливості їхніх терміналів, указує, у якому режимі (з використанням яких кодеків) учасники конференції можуть передавати інформацію, причому цей режим може змінюватися в ході конференції, наприклад, при підключенні до неї нового учасника. Таким чином, контролер MC визначає режим конференції, що може бути загальним для всіх учасників конференції або окремим для кожного з них. В зв’язку з тим, що в мережі може бути кілька контролерів MC, то для кожної знову створюваної конференції повинна проводитися процедура визначення встаткування, для того, щоб виявити той з контролерів MC, що буде управляти даною конференцією. При організації

централізованої конференції, крім контролера МС, повинен використовуватися процесор МР, що обробляє користувальницьку інформацію й відповідає за перемикання або змішування мовних потоків, відеоінформації й даних. При організації децентралізованої конференції процесор МР не використовується.

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

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

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

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

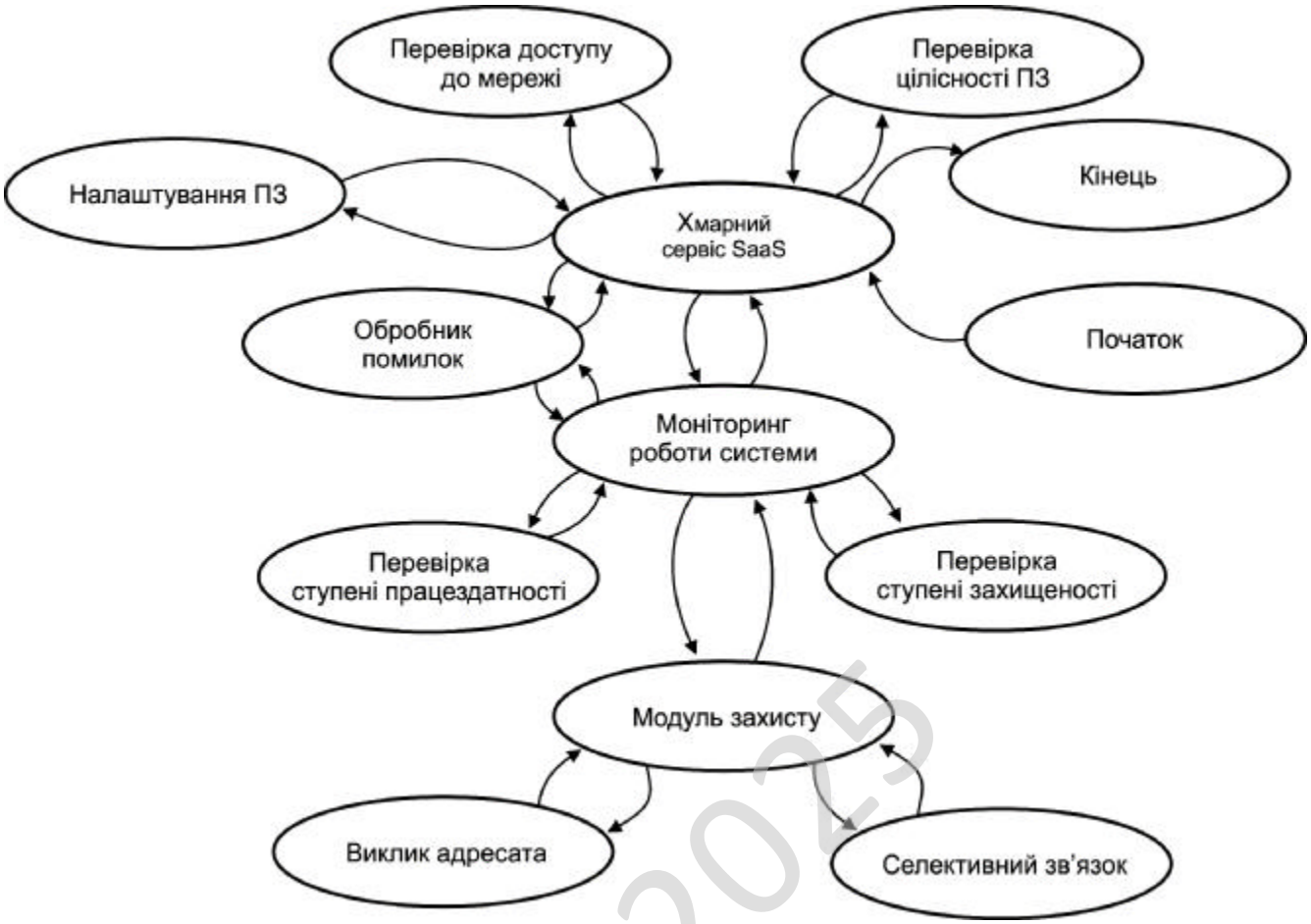

Рисунок 3.3 – Діаграма взаємодії процесів

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Для передачі даних хмарного сервісу SaaS для реалізації віртуальної АТМ по IP-каналам необхідно, як було відмічено раніше, перетворити аналоговий сигнал у цифровий, тобто безперервний в дискретний. Виявляється, що навіть такі на перший погляд зовсім різні сигнали, як безперервні й дискретизовані мають дуже багато загального й зв'язані твердою функціональною залежністю, встановленою теоремою дискретизації, або теоремою Котельникова.

Реальний безперервний сигнал володіє спектром, основна частина енергії якого зосереджена в обмеженій смузі частот. Це зумовлене тим, що прилади, що формують і перетворюють повідомлення і сигнали, а також канали зв'язку мають кінцеву смугу пропускання. Функція часу з обмеженим по ширині спектром повністю визначається своїми миттєвими значеннями, відрахованими через інтервали часу:

$$\Delta t = 1/2F_m, \qquad (4.1)$$

де F_m – найвища частота спектру сигналу.

Це положення складає зміст теореми Котельникова

Теорема дискретизації, або, як її ще називають, теорема Котельникова, теорема Уїтекера, формулюється в такий спосіб: безперервна функція $X(t)$ з обмеженим спектром, тобто не має у своєму спектрі

$$F\{X(t)\} = \int_{-\infty}^{\infty} X(t) \cdot e^{-j2\pi ft}\, dt \qquad (4.2)$$

складових із частотами, що лежать за межами смуги $f \in (-F_m , F_m)$, повністю визначається послідовністю своїх обчислень у дискретні моменти часу $X(t_i)$, що

виходять із кроком $\Delta t < 1/F_m$. Іншими словами безперервна детермінована функція часу $X(t)$, що має обмежений спектр, може бути розкладена в ряд по ортогональним функціям часу виду:

$$\Psi_i(t) = \frac{\sin 2F_m \pi (t - i\Delta t)}{2\pi F_m (t - i\Delta t)} \qquad (4.3)$$

з коефіцієнтами, рівними значенням функції $X(i\Delta t)$. Цей розклад, що називається рядом Котельникова, має наступний вигляд: $X(t) = \sum_{i=-\infty}^{\infty} X(i\Delta t)\Psi_i(t)$.

Функція відліків має наступні властивості:

1. Вона дорівнює 1 при $t = i\Delta t$ і рівна 0 в усіх інших точках $t = j\Delta t; j \neq i$

2. Її спектр рівномірний в межах смуги частот від $-F_{min}$ до F_{max} але рівний 0 поза цією смугою.

3.Вона має властивості ортогональності:

$$\int_{-\infty}^{\infty} \Psi_i(t)\Psi_j(t)\,dt = \begin{cases} 0 & \text{при } i \neq j \\ \dfrac{1}{2F_{max}} & \text{при } i = j \end{cases}$$

Якщо функція f(t) з обмеженим спектром розглядається на кінцевому інтервалі часу T, то число дискрет, що передаються, буде дорівнювати: $T/\Delta t = 2F_m T$. Величину $F_m T$ називають базою сигналу.

Доказ сформульованої теореми ґрунтується на однозначній відповідності між сигналами й відповідними їм спектрами. Іншими словами, якщо сигнали однакові, те й відповідні їм спектри також однакові. І, навпаки, якщо спектри двох сигналів однакові, той і відповідний сигнали також однакові.

Технологічний процес складається з двох основних етапів – збір і облік даних по IP-адресам й формування відповідної легенди та власне сама робота програми у режимі IP-телефонії та селективного зв'язку. Вони можуть виконуватися в будь-який календарний момент часу і включають операції введення, з'єднання і ін. Операції мають програмне виконання, підлегле єдиній алгоритмічній схемі. Програма реалізована в середовищі Delphi.

Робота з програмою починається з виведення інформаційного вікна й активізації системи меню й здійснюється в діалоговому і по-дійному режиму. При цьому під діалогом розуміється надання користувачу декількох альтернатив і обробка його вибору. У діалогову систему входять головне меню з відповідними спливаючими підменю а також діалогові вікна. Під подіями розуміються процеси, що активізуються користувачем (наприклад – натискання функціональних клавіш), а також програмні події – з'єднання по IP-технології.

Програма складається з наступних основних модулів.

Основна процедура – конфігурація середовища оточення, формування основного екрана програми, створення системи головного меню і відповідних підміню, активізація меню.

Процедура обробки головного меню – запуск відповідної процедури. Процедура введення даних – забезпечення введення інформації у бази даних IP-адрес та легенд до них, контроль за допустимістю значень, забезпечення введення даних шляхом вибору зі списку.

Допоміжні процедури і функції – реалізація запитів, повідомлень, формування списків вибору IP-адрес та легенд, а також контроль за даними, що вводяться.

Усі модулі в програмі зв'язані між собою за даними, що аналізуються на вході і виробляються на виході. Дані в модулі надходять через діалог з користувачем, параметри і документи інформаційної бази. Передача даних від одного модуля до іншого здійснюється тільки через збережені документи.

Дані через діалог можуть бути отримані прямим і непрямим способом. Прямий спосіб реалізується шляхом їхнього введення за шаблоном чи по запиту конкретних значень реквізитів. Непрямий спосіб – шляхом чи меню логічних (альтернативних) запитів – «так», «ні». При непрямому способі дані, що надходять у модуль, заздалегідь передбачені алгоритмом, але зовні виглядають в обліку відомими фразами.

Параметри (мова) – вхідні дані, отримані у виді конкретних значень, переданих в оперативній пам'яті суміжним модулям (функціям).

Розглянемо перелік первісних даних. Під вхідною інформацією розуміється вся інформація, необхідна для вирішення задачі і розташована на різних носіях: первинних документах, машинних носіях, у пам'яті персонального комп'ютера. Вхідною інформацією для розроблювальної в проекті корпоративної системи зв’язку з використанням IP-технологій є мова або відеозображення.

Щоб передати мову через телефонну мережу, мовну інформацію потрібно перетворити в аналоговий електричний сигнал. При переході до цифрових мереж зв'язку виникла необхідність перетворити аналоговий електричний сигнал у цифровий формат на передавальній стороні, тобто закодувати, і перевести назад в аналогову форму, тобто декодувати, на прийомній стороні.

Процес перетворення аналогового мовного сигналу в цифрову форму називають аналізом або цифровим кодуванням мови, а зворотний процес відновлення аналогової форми мовного сигналу – синтезом або декодуванням мови.

Ціль будь-якої схеми кодування – одержати таку цифрову послідовність, що вимагає мінімальної швидкості передачі й з якої декодер може відновити вихідний мовний сигнал з мінімальними змінами.

При перетворенні мовного сигналу в цифрову форму, мають місце два процеси – дискретизація, тобто формування дискретних у часі відліків амплітуди сигналу, і квантування, тобто дискретизація отриманих значень за амплітудою. Ці дві функції виконуються т.зв. аналого-цифровими перетворювачами (АЦП), які розміщаються в сучасних АТС на платі абонентських комплектів, а у випадку передачі мови по IP-мережах – у терміналі користувача (комп'ютері або IP-телефоні).

Процес аналого-цифрового перетворення одержав, стосовно до систем зв'язку, назву імпульсно-кодової модуляції (ІКМ).

Щоб знизити необхідну швидкість передачі біт, застосовують нелінійний

(логарифмічний) закон квантування, тобто квантуванню піддається не амплітуда сигналу, а її логарифм. У цьому випадку має місце процес «стиску» динамічного діапазону сигналу, а при відновленні сигналу відбувається зворотний процес.

Сьогодні застосовуються два основні різновиди ІКМ: з кодуванням по m-закону й по А-закону. У результаті стиску сигнал з амплітудою, що кодується 12-13 бітами, описується всього вісьма бітами. Розрізняються ці різновиди ІКМ деталями процесу стиску (m-закон кодування переважніше використовувати при малій амплітуді сигналу й при малому відношенні сигнал/шум). У Північній Америці використовується кодування по m-закону, а в Європі – по А-Закону. Тому при міжнародному зв'язку в багатьох випадках потрібне перетворення m-закону в А-закон, відповідальність за яке несе країна, у якій використовується m-закон кодування. В обох випадках кожний відлік кодується 8 бітами, або одним байтом, який можна вважати звуковим фрагментом. Для передачі послідовності таких фрагментів необхідна пропускна здатність каналу, рівна 64 Кбіт/с. Оскільки ІКМ була першою стандартною технологією, що одержала широке застосування в цифрових системах передачі, пропускна здатність каналу, рівна 64 Кбіт/с, стала всесвітнім стандартом для цифрових мереж всіх видів, причому – стандартом, що забезпечує передачу мови з дуже гарною якістю. Однак така висока якість передачі мовного сигналу (що є еталоном при оцінці якості інших схем кодування) досягнута в системах ІКМ за рахунок явно надлишкова, при сучасному рівні технології, швидкості передачі інформації.

Щоб зменшити властиву ІКМ надмірність і знизити вимоги до смуги пропущення, послідовність чисел, отримана в результаті перетворення мовного аналогового сигналу в цифрову форму, піддається математичним перетворенням, що дозволяють зменшити необхідну швидкість передачі. Ці перетворення «сирого» цифрового потоку в потік меншої швидкості називають «стиском» (а часто – кодуванням, розглядаючи ІКМ як якусь відправну точку для подальшої обробки інформації). Існує безліч підходів до «стиску» мовної інформації; всі їх можна розділити на три категорії: кодування форми сигналу (waveform coding),

кодування вихідної інформації (source coding) і гібридне кодування, що представляє собою сполучення двох підходів.

Кодування форми сигналу – імпульсно-кодова модуляція, по суті, і являє собою схему кодування форми сигналу. Однак цікавлять більш складні алгоритми, що дозволяють знизити вимоги до смуги пропущення. Розглянуті методи кодування форми сигналу використовують ту обставину, що між випадковими значеннями декількох послідовних обчислень існує деяка залежність. Це дозволяє з досить високою точністю пророчити значення будь-якого відліку на основі значень декількох попередніх йому обчислень.

При побудові алгоритмів кодування названа закономірність використовується двома способами. По-перше, є можливість змінювати параметри квантування залежно від характеру сигналу. У цьому випадку крок квантування може змінюватися, що дозволяє певною мірою згладити протиріччя між зменшенням числа біт, необхідних для кодування величини відліку при збільшенні кроку квантування, і звуженням динамічного діапазону кодера, неминучим без адаптації. Деякі алгоритми передбачають зміну параметрів квантування приблизно в рамках вимовних складів, а деякі змінюють крок квантування на основі аналізу статистичних даних про амплітуду сигналу, отриманих за відносно короткий проміжок часу. По-друге, існує підхід, називаний диференціальним кодуванням або лінійним пророкуванням. Замість того, щоб кодувати вхідний сигнал безпосередньо, кодують різницю між вхідним сигналом і «передвіщеною» величиною, обчисленою на основі декількох попередніх значень сигналу.

Описаний метод називається лінійним пророкуванням, тому що він використовує тільки лінійні функції попередніх обчислень. Найпростішою реалізацією останнього підходу є так звана дельта-модуляція (ДМ), алгоритм якої передбачає кодування різниці між сусідніми обчисленнями сигналу тільки одним інформаційним бітом, забезпечуючи передачу, по суті, тільки знака різниці.

Алгоритмом, побудованим на описані вище принципах, є алгоритм

адаптивної диференціальної імпульсно-кодової модуляції (АДІКМ) (G.726). Алгоритм передбачає формування сигналу помилки пророкування і його наступне адаптивне квантування. При досить гарних характеристиках алгоритму, АДІКМ практично не застосовується для передачі мови по мережах з комутацією пакетів, тому що цей алгоритм дуже чутливий до втрат цілих блоків відліку, що відбуваються при втратах пакетів у мережі. У таких випадках порушується синхронізація кодера й декодера, що приводить до катастрофічного погіршення якості відтворення мови навіть при малій імовірності втрат.

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

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

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю хмарного сервісу SaaS для реалізації віртуальної АТМ.

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

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних

аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

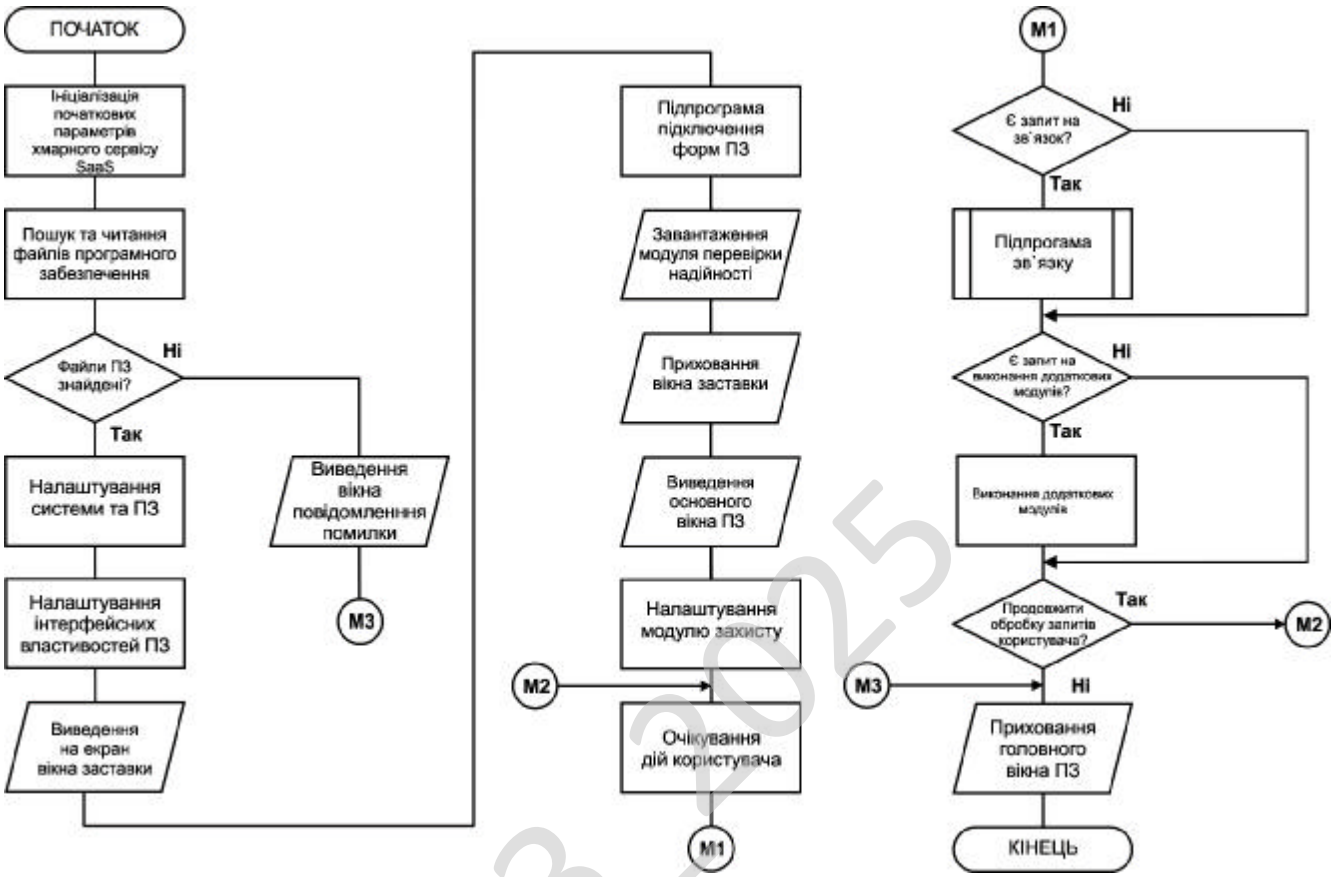

Рисунок 4.1 – Блок-схема основної програми

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

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

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще

більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

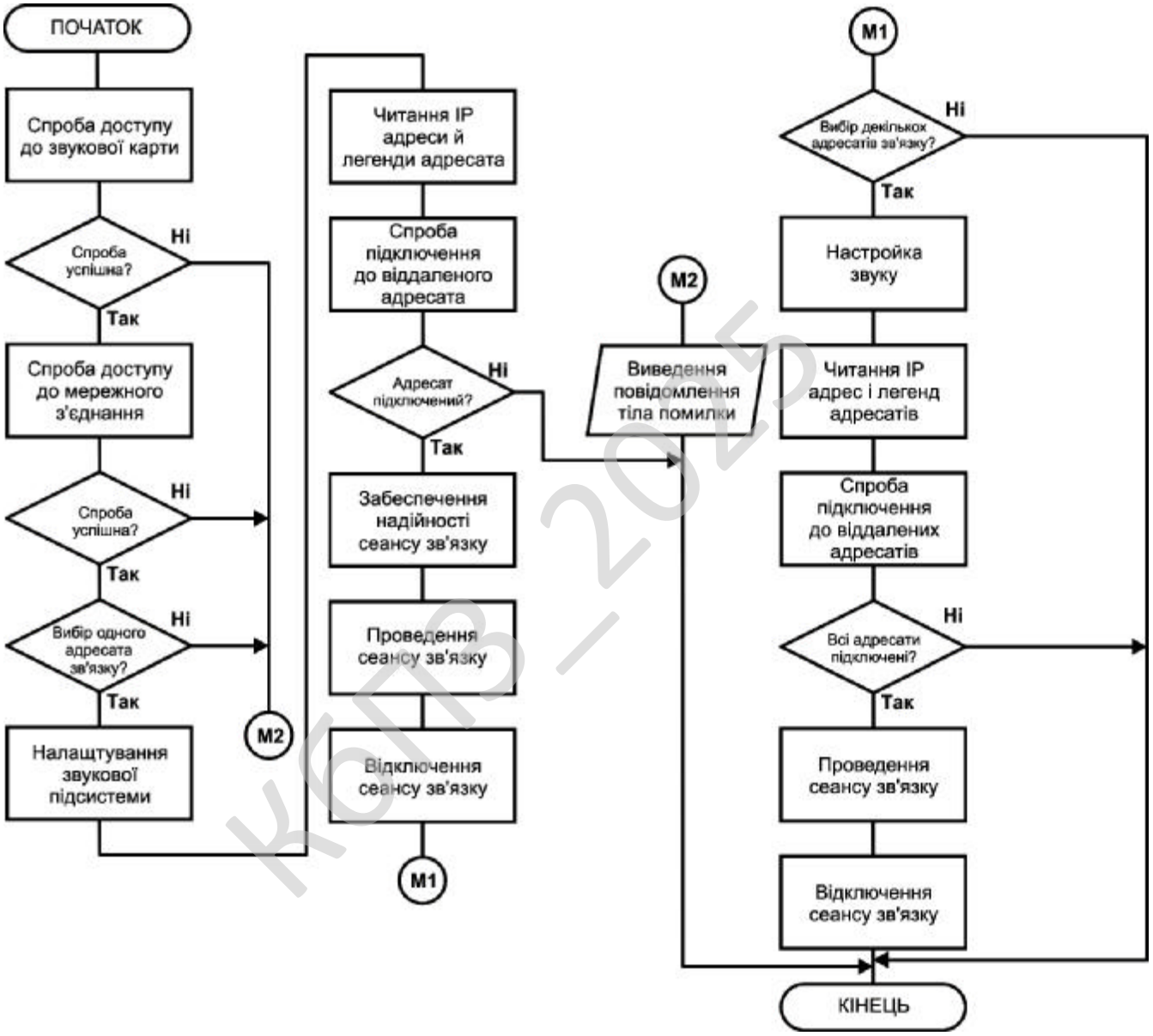

Рисунок 4.2 – Блок-схема роботи підпрограми

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

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

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

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

– Програмістам які реалізовують модулі інформаційної системи.

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

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу).

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою RC6 – симетричний блоковий криптографічний алгоритм, похідний від алгоритму RC5. Був створений Роном Рівестом, Меттом Робшау і Реєм Сіднеєм для задоволення вимог конкурсу Advanced Encryption Standard (AES). Алгоритм був одним з п'яти фіналістів конкурсу, був також представлений NESSIE і CRYPTREC. Є власницьким (пропрієтарним) алгоритмом, і запатентований RSA Security, одначе дія патентів сплила, і зараз алгоритм знаходиться у відкритому доступі. В той же час, "RC6" залишається зареєстрованою торговою маркою RSA.

Варіант шифру RC6, заявлений на конкурс AES, підтримує блоки довжиною 128 біт і ключі довжиною 128, 192 і 256 біт, але сам алгоритм, як і RC5, може бути налаштований для підтримки більш широкого діапазону довжин як блоків, так і ключів (від 0 до 2040 біт)[1]. RC6 дуже схожий на RC5 за своєю структурою і також досить простий у реалізації.

Є фіналістом AES, проте одна з примітивних операцій – операція множення, повільно виконується на певному обладнанні і ускладнює реалізацію шифру на ряді апаратних платформ і, що виявилося сюрпризом для авторів, на системах з архітектурою Intel IA-64 також реалізована досить погано. В даному випадку алгоритм втрачає одну зі своїх ключових переваг – високу швидкість виконання, що стало причиною для критики і однією з перепон для обрання як нового стандарту.

Деталі RC6

Так само, як і RC5, RC6 – повністю параметризована сім'я алгоритмів шифрування. Для специфікації алгоритму з конкретними параметрами, прийнято позначення RC6-w/r/b, де

– W – довжина машинного слова в бітах.

– R – число раундів.

– B – довжина ключа в байтах. Можливі значення 0 .. 255 байт.

Для того щоб відповідати вимогам AES, блочний шифр повинен працювати з 128-бітовими блоками. Так як RC5 – виключно швидкий блочний шифр, розширення його, щоб працювати з 128-бітовими блоками, привело б до використання двох 64-бітових робочих регістрів. Але архітектура і мови програмування ще не підтримують 64-бітні операції, тому довелося змінити проект так, щоб використовувати чотири 32-бітних регістри замість двох 64-бітних.

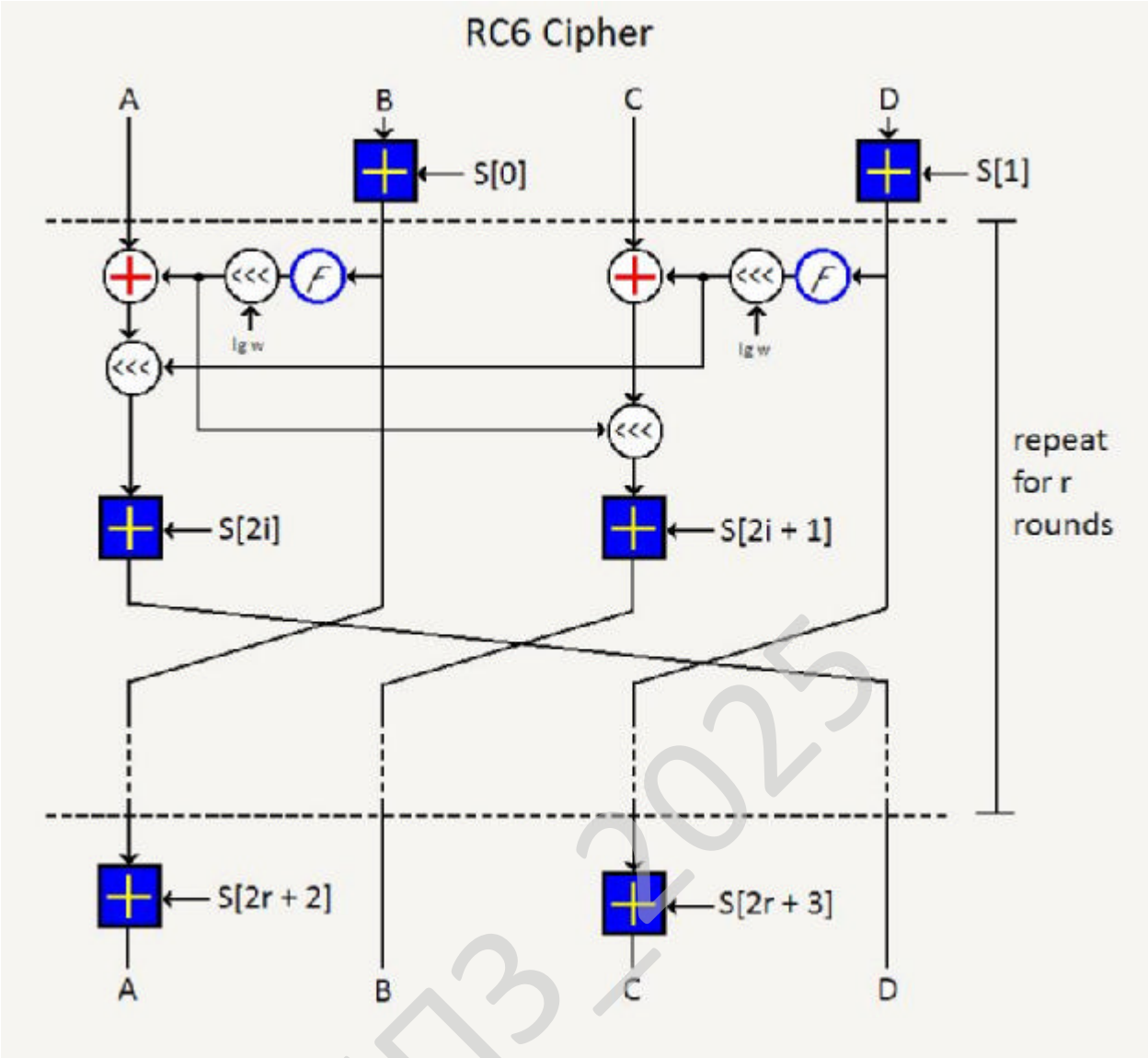

Рисунок 4.3 – Структура RC6

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ хмарного сервісу SaaS для реалізації віртуальної АТМ яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Сеанс зв’язку; Дані АТМ; Налаштування.

– Розділу обрання групи запису.

– Розділу виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ хмарного сервісу SaaS для реалізації віртуальної АТМ.

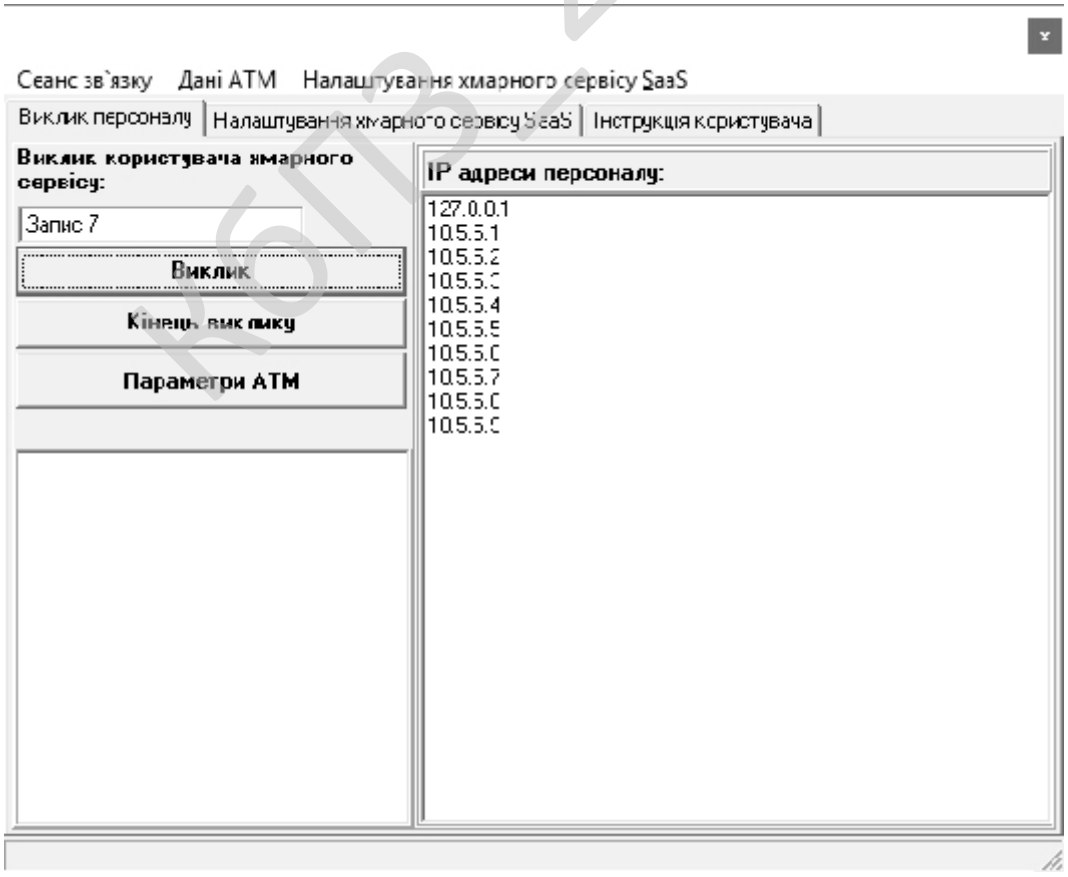

Рисунок 5.1 – Головне вікно ПЗ

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

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

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

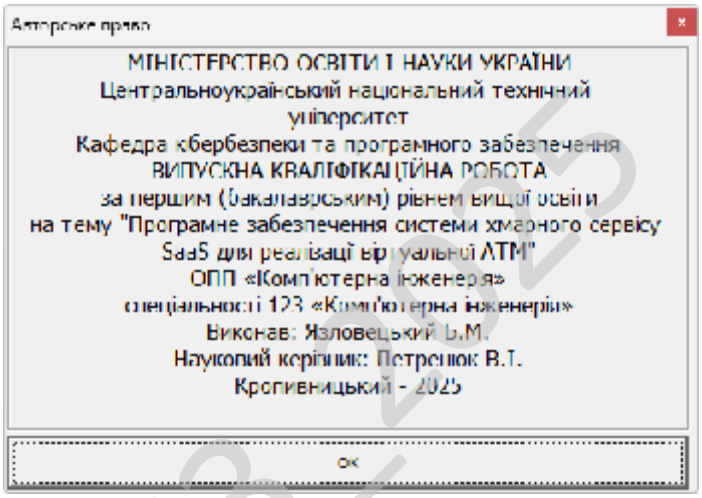

Рисунок 5.2 – Авторське право

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

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити.

Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

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

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

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в ІТ рішення за принципом найбільшої корисності для більшості учасників.

Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

Проводилось тестування чорної скриньки. Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

– Помилок характеристик (необхідна ємність пам'яті і т.д.).

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем хмарного сервісу SaaS для реалізації віртуальної АТМ.

– Досліджена система хмарного сервісу SaaS для реалізації віртуальної АТМ.

– На основі отриманих результатів досліджень створена програмна реалізація системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання хмарного сервісу SaaS для реалізації віртуальної АТМ.

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

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

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи хмарного сервісу SaaS для реалізації віртуальної АТМ. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на

загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм RC6.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kopf, Johannes and Lischinski, Dani. Depixelizing Pixel Art (англ.) // ACM Trans. Graph. – 2011. – Vol. 30, no. 4. – P. 99:1--99:8.

2. Giachetti, Andrea and Asuni, Nicola. Real-Time Artifact-Free Image Upscaling (англ.) // Trans. Img. Proc.. – 2011. – Vol. 20, no. 10. – P. 2760—2768.

3. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

4. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447

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

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

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

8. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» CEUR Workshop Proceedings, Volume 3187, 2022,

9. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.

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

11. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021, Cracow, Poland, 22-25 September 2021. P. 414-418.

12. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». 4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

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

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

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

16. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». CEUR Workshop Proceedings Volume 2654, 2020, Pages 1-14.

17. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

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

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

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

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

22. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», CEUR Workshop Proceedings Volume 2608, 2020, Pages 646-660.

23. Zhurakovskyi, B., Tsopa, N., Batrak, Y., Odarchenko, R., Smirnova, T «Comparative analysis of modern formats of lossy audio compression». Workshop Proceedings, 2020, 2654, стр. 315-327.

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

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

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

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

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

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

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

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

32. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated Pseudorandom Sequences». CEUR Workshop Proceedings Volume 2732, 2020, Pages 214-227.

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

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

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

36. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов «Хмарна інформаційна система оцінювання шорсткості з використанням дискретного частотного аналізу макрофотографій». IV міжнародна науково-практична конференція “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 15-16 квітня 2021р. – Кропивницький: ЦНТУ. – 2021. – С. 30.

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

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

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

40. О. Смірнов, Є. Деменко, О. Онікійчук, А. Арищенко, Л. Горбачова, «Формування псевдовипадкових послідовностей для приховування даних в зображеннях» Комп'ютерні науки та кібербезпека. № 4. С. 30-37. 2019.

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

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

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

44. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для

моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

45. Смірнов О.А., Кавун С.В., Коваленко О.В., Дрєєв О.М. Мережні інформаційні технології. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 159 с.

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

47. Смірнов О.А., Кавун С.В., Коваленко О.В., Доренський О.П., Дрєєв О.М., Вялкова В.І. Комп'ютерні мережі. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 233 с.

48. Смірнов О.А., Дрєєв О.М. Порівняння бітових щільностей при використанні різних методів кодування інформації. Збірник наукових праць "Системи обробки інформації". - Випуск 2 (118). т.2. - Х.: ХУПС - 2014. - С. 64-67

49. Смірнов О.А., Дрєєв О.М. Порівняння бітових щільностей при використанні різних методів кодування інформації. Збірник тез VI міжнародної науково-практичної конференції "Проблеми та перспективи розвитку ІТ-індустрії". м. Харків. 17-18 квітня 2014р. – Харків: ХНЄУ. - 2014. - С. 240.

50. Смірнов О.А., Коваленко О.В., Кожанова А.С., Левошко О.Л., Константинова Л.В. Основи системного програмування. Навчальний посібник. – Кіровоград: КНТУ 2013. – 257с.

51. Смірнов О.А., Дрєєв О.М., Доренський О.П. «Дослідження впливу ступеня стиснення зображень на оперативність їх доставки у телекомунікаційній системі. Збірник наукових праць "Системи обробки інформації". – Випуск 8(115). – Х.: ХУПС – 2013. – С. 234-239.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

2 Підстава для розробки

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

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи хмарного сервісу SaaS для реалізації віртуальної АТМ;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 86 аркушів.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

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

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

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Петренюк В.І.

Програмне забезпечення системи хмарного сервісу SaaS для реалізації віртуальної АТМ

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 24

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
#реалізує систему хмарного сервісу SaaS для віртуальної АТМ
#Система надає можливості авторизації, перегляду балансу, депозиту,
# зняття коштів та переказів
#Код написаний на Python з використанням фреймворку Flask для створення REST API
#Кожна функція містить детальні коментарі для роз'яснення її роботи
#Система використовує внутрішні сховища даних для симуляції бази даних
# користувачів та сесій

import uuid
import datetime
import time
import threading
from flask import Flask, request, jsonify

#Ініціалізація Flask додатку
app = Flask(__name__)

#Глобальний словник для зберігання даних облікових записів користувачів
accounts = {}

#Глобальний словник для зберігання сесій користувачів
sessions = {}

#Глобальний список для зберігання журналу транзакцій
transaction_logs = []

#Функція ініціалізації демонстраційних даних облікових записів
def initialize_dummy_data():
#Функція створює декілька демонстраційних користувачів з початковими балансами
та даними для входу
    global accounts
    accounts["user1"] = {
        "username": "user1",
        "password": "pass1",
        "balance": 1000.0,
        "transactions": []
    }
#Створення другого облікового запису для користувача user2
    accounts["user2"] = {
        "username": "user2",
        "password": "pass2",
        "balance": 2000.0,
        "transactions": []
    }
#Створення третього облікового запису для користувача user3
    accounts["user3"] = {
        "username": "user3",
        "password": "pass3",
        "balance": 3000.0,
        "transactions": []
    }
#Кінець ініціалізації демонстраційних даних

#Функція генерації унікального токену для сесії користувача
def generate_token():
#Генеруємо токен використовуючи бібліотеку uuid для забезпечення унікальності
    return str(uuid.uuid4())
#Кінець функції генерації токену

#Функція для запису інформації про транзакції в журнал
```

```
def log_transaction(username, transaction_type, amount):
#Функція записує тип транзакції, суму та час проведення операції
    timestamp = datetime.datetime.now().isoformat()
    log_entry = {
        "username": username,
        "transaction_type": transaction_type,
        "amount": amount,
        "timestamp": timestamp
    }
    transaction_logs.append(log_entry)
#Кінець функції журналювання транзакцій

#Функція симуляції затримки виклику зовнішнього API
def simulate_external_api_call():
#Функція імітує затримку, яка може виникати при зверненні до зовнішніх сервісів
у хмарному середовищі
    time.sleep(0.1)
#Кінець функції симуляції зовнішнього виклику

#Функція аутентифікації користувача за вхідними даними
def authenticate_user(username, password):
#Перевіряємо чи існує користувач та чи відповідає введений пароль збереженому
    if username in accounts:
        if accounts[username]["password"] == password:
            return True
    return False
#Кінець функції аутентифікації користувача

#Функція для отримання даних облікового запису за токеном сесії
def get_account_by_token(token):
#За допомогою токену визначаємо користувача, пов’язаний з поточною сесією
    if token in sessions:
        username = sessions[token]
        if username in accounts:
            return accounts[username]
    return None
#Кінець функції отримання облікового запису за токеном

#HTTP-ендпоінт для входу користувача в систему
@app.route("/login", methods=["POST"])
def login():
#Ендпоінт приймає JSON дані з полями username та password
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    username = data.get("username")
    password = data.get("password")
    simulate_external_api_call()
    if authenticate_user(username, password):
        token = generate_token()
        sessions[token] = username
        return jsonify({"message": "Login successful", "token": token}), 200
    else:
        return jsonify({"error": "Invalid credentials"}), 401
#Кінець ендпоінту входу

#HTTP-ендпоінт для виходу користувача з системи
@app.route("/logout", methods=["POST"])
def logout():
#Ендпоінт приймає JSON дані з токеном для завершення сесії
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
```

```
    token = data.get("token")
    simulate_external_api_call()
    if token in sessions:
        del sessions[token]
        return jsonify({"message": "Logout successful"}), 200
    else:
        return jsonify({"error": "Invalid token"}), 401
#Кінець ендпоінту виходу

#HTTP-ендпоінт для отримання балансу користувача
@app.route("/balance", methods=["GET"])
def balance():
#Ендпоінт перевіряє заголовок Authorization для отримання токену сесії
    token = request.headers.get("Authorization")
    simulate_external_api_call()
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
    return jsonify({"balance": account["balance"]}), 200
#Кінець ендпоінту перегляду балансу

#HTTP-ендпоінт для депозиту коштів на рахунок
@app.route("/deposit", methods=["POST"])
def deposit():
#Ендпоінт приймає JSON дані з токеном та сумою депозиту
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    amount = data.get("amount")
    simulate_external_api_call()
    if not isinstance(amount, (int, float)) or amount <= 0:
        return jsonify({"error": "Invalid deposit amount"}), 400
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    account["balance"] += amount
    account["transactions"].append({
        "type": "deposit",
        "amount": amount,
        "timestamp": datetime.datetime.now().isoformat()
    })
    log_transaction(account["username"], "deposit", amount)
    return jsonify({"message": "Deposit successful", "new_balance":
account["balance"]}), 200
#Кінець ендпоінту депозиту коштів

#HTTP-ендпоінт для зняття коштів з рахунку
@app.route("/withdraw", methods=["POST"])
def withdraw():
#Ендпоінт приймає JSON дані з токеном та сумою зняття коштів
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    amount = data.get("amount")
    simulate_external_api_call()
    if not isinstance(amount, (int, float)) or amount <= 0:
        return jsonify({"error": "Invalid withdrawal amount"}), 400
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    if account["balance"] < amount:
```

```
        return jsonify({"error": "Insufficient funds"}), 400
    account["balance"] -= amount
    account["transactions"].append({
        "type": "withdraw",
        "amount": amount,
        "timestamp": datetime.datetime.now().isoformat()
    })
    log_transaction(account["username"], "withdraw", amount)
    return jsonify({"message": "Withdrawal successful", "new_balance":
account["balance"]}), 200
#Кінець ендпоінту зняття коштів

#HTTP-ендпоінт для переказу коштів між рахунками
@app.route("/transfer", methods=["POST"])
def transfer():
#Ендпоінт приймає JSON дані з токеном, цільовим ім'ям користувача та сумою
переказу
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    target_username = data.get("target_username")
    amount = data.get("amount")
    simulate_external_api_call()
    if not isinstance(amount, (int, float)) or amount <= 0:
        return jsonify({"error": "Invalid transfer amount"}), 400
    source_account = get_account_by_token(token)
    if source_account is None:
        return jsonify({"error": "Invalid token"}), 401
    if source_account["balance"] < amount:
        return jsonify({"error": "Insufficient funds"}), 400
    if target_username not in accounts:
        return jsonify({"error": "Target account does not exist"}), 400
    source_account["balance"] -= amount
    accounts[target_username]["balance"] += amount
    timestamp = datetime.datetime.now().isoformat()
    source_account["transactions"].append({
        "type": "transfer_out",
        "amount": amount,
        "target": target_username,
        "timestamp": timestamp
    })
    accounts[target_username]["transactions"].append({
        "type": "transfer_in",
        "amount": amount,
        "source": source_account["username"],
        "timestamp": timestamp
    })
    log_transaction(source_account["username"], "transfer_out", amount)
    log_transaction(target_username, "transfer_in", amount)
    return jsonify({"message": "Transfer successful", "new_balance":
source_account["balance"]}), 200
#Кінець ендпоінту переказу коштів

#HTTP-ендпоінт для отримання історії транзакцій користувача
@app.route("/transactions", methods=["GET"])
def transactions():
#Ендпоінт повертає список усіх транзакцій, здійснених користувачем
    token = request.headers.get("Authorization")
    simulate_external_api_call()
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
```

```
    return jsonify({"transactions": account["transactions"]}), 200
#Кінець ендпоінту історії транзакцій

#Функція для виконання перевірки стану системи
def system_health_check():
#Функція повертає статус системи, час перевірки, кількість активних сесій та
кількість зареєстрованих облікових записів
    health_status = {
        "status": "OK",
        "time": datetime.datetime.now().isoformat(),
        "active_sessions": len(sessions),
        "registered_accounts": len(accounts)
    }
    return health_status
#Кінець функції перевірки стану системи

#HTTP-ендпоінт для перевірки стану системи
@app.route("/health", methods=["GET"])
def health():
#Ендпоінт повертає базову інформацію про роботу системи для моніторингу
    simulate_external_api_call()
    health_status = system_health_check()
    return jsonify(health_status), 200
#Кінець ендпоінту перевірки стану системи

#Додаткова функція для симуляції фонової обробки завдань у системі
def background_job_processing():
#Функція імітує виконання періодичних фоновых завдань, наприклад, обробку черг
транзакцій
    while True:
        simulate_external_api_call()
#Імітуємо обробку транзакцій (демонстраційний приклад)
        time.sleep(5)
#Кінець циклу фонового процесу

#Додаткова функція для симуляції планових завдань технічного обслуговування
def scheduled_tasks():
#Функція виконує планові завдання, такі як логування поточного стану системи та
інші профілактичні операції
    while True:
        simulate_external_api_call()
#Вивід поточного статусу системи для моніторингу
        current_status = system_health_check()
        print("Scheduled Task - System Status:", current_status)
        time.sleep(10)
#Кінець циклу планових завдань

#Основний блок виконання програми
if __name__ == "__main__":
#Ініціалізація демонстраційних даних облікових записів
    initialize_dummy_data()
#Вивід повідомлення про запуск системи
    print("Virtual ATM Cloud Service is starting up...")
#Створення окремого потоку для фонового процесу обробки завдань
    bg_thread = threading.Thread(target=background_job_processing, daemon=True)
    bg_thread.start()
#Створення окремого потоку для виконання планових завдань
    scheduled_thread = threading.Thread(target=scheduled_tasks, daemon=True)
    scheduled_thread.start()
#Запуск Flask додатку на порту 5000
    app.run(host="0.0.0.0", port=5000, debug=False)
#Кінець основного блоку виконання програми
```

#Кінець реалізації системи хмарного сервісу SaaS для віртуальної АТМ

Файл create_genesis_block.py

```
#!/usr/bin/env python3
#Цей скрипт реалізує систему хмарного сервісу SaaS для віртуальної АТМ
#Система надає можливості авторизації, реєстрації,
# багаторівневої аутентифікації,
#перегляду балансу, депозиту, зняття коштів, переказів, а також розширену
логіку:
#1. Реєстрація користувачів з підтвердженням електронною поштою.
#2. Многофакторна автентифікація (MFA) через OTP.
#3. Нотифікаційна система в реальному часі для повідомлення користувачів.
#4. Блокчейн-реєстр транзакцій для забезпечення прозорості.
#5. Розширене логування та аудит дій користувачів.
#
#Код написаний на Python з використанням Flask для створення REST API

import uuid
import datetime
import time
import threading
import hashlib
import random
import string
import os
from flask import Flask, request, jsonify

#Ініціалізація Flask додатку
app = Flask(__name__)

#Глобальний словник для зберігання даних облікових записів користувачів
accounts = {}

#Глобальний словник для зберігання сесій користувачів
sessions = {}

#Глобальний список для зберігання журналу транзакцій
transaction_logs = []

#Глобовий словник для зберігання тимчасових реєстраційних даних
registration_tokens = {}

#Глобовий словник для зберігання OTP кодів для MFA
otp_codes = {}

#Глобовий словник для зберігання нотифікацій користувачів
notifications = {}

#Глобовий список для зберігання аудит-логів
audit_logs = []

#Глобовий список, який представляє блокчейн-реєстр транзакцій
blockchain = []

#------------------ ФУНКЦІЇ ДЛЯ РЕЄСТРАЦІЇ КОРИСТУВАЧІВ ----------------------#

#Функція для генерації унікального токену (реєстраційного чи сесійного)
def generate_token():
    #Генеруємо токен використовуючи UUID для забезпечення унікальності
    return str(uuid.uuid4())
#Кінець функції генерації токену

#Функція для генерації OTP коду для MFA
def generate_otp_code(length=6):
```

```
    #Генеруємо випадковий числовий OTP код заданої довжини
    otp = ''.join(random.choices(string.digits, k=length))
    return otp
#Кінець функції генерації OTP коду

#Функція для додавання нового повідомлення у нотифікаційну систему
def add_notification(username, message):
    #Перевіряємо чи існує запис для користувача, якщо ні, створюємо новий список
    if username not in notifications:
        notifications[username] = []
    notifications[username].append({
        "message": message,
        "timestamp": datetime.datetime.now().isoformat()
    })
#Кінець функції додавання нотифікації

#Функція для запису аудит-логів у глобальний список та файл
def audit_log(event):
    #Отримуємо поточний час для логування події
    timestamp = datetime.datetime.now().isoformat()
    log_entry = {"event": event, "timestamp": timestamp}
    audit_logs.append(log_entry)
    #Записуємо лог в файл для довгострокового зберігання
    with open("audit.log", "a", encoding="utf-8") as f:
        f.write(f"{timestamp} - {event}\n")
#Кінець функції аудит-логування

#---------------------- ФУНКЦІЇ ДЛЯ БЛОКЧЕЙН РЕЄСТРУ ----------------------#

#Функція для обчислення хешу блоку за допомогою SHA-256
def calculate_hash(index, timestamp, transactions, previous_hash):
    #Створюємо рядок даних для хешування
    block_string = f"{index}{timestamp}{transactions}{previous_hash}"
    #Обчислюємо хеш використовуючи hashlib
    return hashlib.sha256(block_string.encode()).hexdigest()
#Кінець функції обчислення хешу

#Функція для створення генезис блоку (перший блок у блокчейні)
def create_genesis_block():
    #Ініціалізуємо генезис блок з фіксованими значеннями
    index = 0
    timestamp = datetime.datetime.now().isoformat()
    transactions_data = "Genesis Block"
    previous_hash = "0"
    hash_val = calculate_hash(index, timestamp, transactions_data,
previous_hash)
    genesis_block = {
        "index": index,
        "timestamp": timestamp,
        "transactions": transactions_data,
        "previous_hash": previous_hash,
        "hash": hash_val
    }
    blockchain.append(genesis_block)
#Кінець функції створення генезис блоку

#Функція для додавання нового блоку до блокчейну
def add_block(transactions_list):
    #Отримуємо останній блок з блокчейну
    last_block = blockchain[-1]
    index = last_block["index"] + 1
    timestamp = datetime.datetime.now().isoformat()
    transactions_data = str(transactions_list)
```

```
    previous_hash = last_block["hash"]
    hash_val = calculate_hash(index, timestamp, transactions_data,
previous_hash)
    new_block = {
        "index": index,
        "timestamp": timestamp,
        "transactions": transactions_list,
        "previous_hash": previous_hash,
        "hash": hash_val
    }
    blockchain.append(new_block)
#Кінець функції додавання блоку

#Функція для інтеграції транзакції в блокчейн
def blockchain_log_transaction(transaction):
    #Додаємо транзакцію в транзакційний журнал блокчейну
    add_block(transaction)
#Кінець функції інтеграції транзакції в блокчейн

#-------------------- ФУНКЦІЇ ДЛЯ РЕЄСТРАЦІЇ ТА MFA ----------------------#

#HTTP-ендпоінт для реєстрації користувача
@app.route("/register", methods=["POST"])
def register():
    #Ендпоінт приймає JSON дані з полями username, password та email
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    username = data.get("username")
    password = data.get("password")
    email = data.get("email")
    #Перевірка наявності всіх необхідних даних
    if not username or not password or not email:
        return jsonify({"error": "Missing username, password or email"}), 400
    #Перевіряємо чи існує користувач із таким ім'ям
    if username in accounts or username in registration_tokens:
        return jsonify({"error": "User already exists or registration
pending"}), 400
    #Генеруємо унікальний токен для підтвердження реєстрації
    token = generate_token()
    registration_tokens[token] = {
        "username": username,
        "password": password,
        "email": email,
        "balance": 0.0,
        "transactions": []
    }
    #Логування події реєстрації
    audit_log(f"Registration initiated for username: {username}")
    #Симулюємо відправлення підтвердження на email користувача
    print(f"Simulated email sent to {email} with confirmation token: {token}")
    return jsonify({"message": "Registration initiated. Please confirm via the
token sent to your email.", "confirmation_token": token}), 200
#Кінець ендпоінту реєстрації

#HTTP-ендпоінт для підтвердження реєстрації користувача
@app.route("/confirm_registration", methods=["POST"])
def confirm_registration():
    #Ендпоінт приймає JSON дані з токеном підтвердження
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("confirmation_token")
```

```
    if token not in registration_tokens:
        return jsonify({"error": "Invalid or expired confirmation token"}), 400
    #Отримуємо дані користувача з тимчасового сховища
    user_data = registration_tokens.pop(token)
    username = user_data["username"]
    #Додаємо користувача до основного сховища облікових записів
    accounts[username] = user_data
    #Ініціалізуємо нотифікації для нового користувача
    notifications[username] = []
    #Логування події підтвердження реєстрації
    audit_log(f"Registration confirmed for username: {username}")
    return jsonify({"message": f"User {username} successfully registered."}),
200
#Кінець ендпоінту підтвердження реєстрації

#HTTP-ендпоінт для ініціалізації MFA (відправка OTP)
@app.route("/send_otp", methods=["POST"])
def send_otp():
    #Ендпоінт приймає JSON дані з токеном сесії
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
    username = account["username"]
    #Генеруємо OTP код
    otp = generate_otp_code()
    #Встановлюємо час дії OTP на 5 хвилин від поточного моменту
    expiry_time = time.time() + 300
    otp_codes[username] = {"otp": otp, "expiry": expiry_time}
    #Симулюємо відправлення OTP на email користувача
    print(f"Simulated OTP sent to {username}'s email: {otp}")
    #Логування події надсилання OTP
    audit_log(f"OTP sent for MFA for username: {username}")
    return jsonify({"message": "OTP sent to your registered email."}), 200
#Кінець ендпоінту надсилання OTP

#HTTP-ендпоінт для перевірки OTP коду (MFA)
@app.route("/mfa_verify", methods=["POST"])
def mfa_verify():
    #Ендпоінт приймає JSON дані з токеном сесії та OTP кодом
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    otp_input = data.get("otp")
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
    username = account["username"]
    if username not in otp_codes:
        return jsonify({"error": "OTP not requested"}), 400
    otp_data = otp_codes.get(username)
    #Перевіряємо чи не минув термін дії OTP
    if time.time() > otp_data["expiry"]:
        otp_codes.pop(username, None)
        return jsonify({"error": "OTP expired"}), 400
    #Перевіряємо правильність введеного OTP
    if otp_input != otp_data["otp"]:
        return jsonify({"error": "Invalid OTP"}), 400
    #Після успішної перевірки видаляємо збережений OTP
```

```
    otp_codes.pop(username, None)
    #Логування події успішної MFA аутентифікації
    audit_log(f"MFA verified successfully for username: {username}")
    #Додаємо сповіщення про успішну MFA
    add_notification(username, "Multi-Factor Authentication completed
successfully.")
    return jsonify({"message": "MFA verification successful."}), 200
#Кінець ендпоінту перевірки MFA




#------------------- ФУНКЦІЇ ДЛЯ ОСНОВНОГО ФУНКЦІОНАЛУ ----------------------#

#Функція ініціалізації демонстраційних даних облікових записів
def initialize_dummy_data():
    #Ініціалізуємо декілька демонстраційних користувачів з початковими балансами
    global accounts
    accounts["user1"] = {
        "username": "user1",
        "password": "pass1",
        "balance": 1000.0,
        "transactions": []
    }
    accounts["user2"] = {
        "username": "user2",
        "password": "pass2",
        "balance": 2000.0,
        "transactions": []
    }
    accounts["user3"] = {
        "username": "user3",
        "password": "pass3",
        "balance": 3000.0,
        "transactions": []
    }
    #Ініціалізуємо нотифікації для демонстраційних користувачів
    notifications["user1"] = []
    notifications["user2"] = []
    notifications["user3"] = []
    #Логування події ініціалізації даних
    audit_log("Dummy data for accounts initialized.")
#Кінець ініціалізації демонстраційних даних

#Функція симуляції затримки виклику зовнішнього API
def simulate_external_api_call():
    #Імітуємо затримку при зверненні до зовнішніх сервісів
    time.sleep(0.1)
#Кінець функції симуляції зовнішнього API

#Функція аутентифікації користувача за вхідними даними
def authenticate_user(username, password):
    #Перевіряємо наявність користувача та відповідність пароля
    if username in accounts:
        if accounts[username]["password"] == password:
            return True
    return False
#Кінець функції аутентифікації

#Функція для отримання даних облікового запису за токеном сесії
def get_account_by_token(token):
    #За допомогою токену визначаємо користувача, пов’язаний з поточною сесією
```

```
    if token in sessions:
        username = sessions[token]
        if username in accounts:
            return accounts[username]
    return None
#Кінець функції отримання облікового запису за токеном

#HTTP-ендпоінт для входу користувача в систему
@app.route("/login", methods=["POST"])
def login():
    #Ендпоінт приймає JSON дані з полями username та password
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    username = data.get("username")
    password = data.get("password")
    simulate_external_api_call()
    if authenticate_user(username, password):
        token = generate_token()
        sessions[token] = username
        #Логування успішного входу
        audit_log(f"User {username} logged in.")
        add_notification(username, "You have successfully logged in.")
        return jsonify({"message": "Login successful", "token": token}), 200
    else:
        audit_log(f"Failed login attempt for username: {username}")
        return jsonify({"error": "Invalid credentials"}), 401
#Кінець ендпоінту входу

#HTTP-ендпоінт для виходу користувача з системи
@app.route("/logout", methods=["POST"])
def logout():
    #Ендпоінт приймає JSON дані з токеном для завершення сесії
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    simulate_external_api_call()
    account = get_account_by_token(token)
    if token in sessions:
        username = sessions[token]
        del sessions[token]
        audit_log(f"User {username} logged out.")
        add_notification(username, "You have successfully logged out.")
        return jsonify({"message": "Logout successful"}), 200
    else:
        return jsonify({"error": "Invalid token"}), 401
#Кінець ендпоінту виходу

#HTTP-ендпоінт для отримання балансу користувача
@app.route("/balance", methods=["GET"])
def balance():
    #Ендпоінт перевіряє заголовок Authorization для отримання токену сесії
    token = request.headers.get("Authorization")
    simulate_external_api_call()
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
    return jsonify({"balance": account["balance"]}), 200
#Кінець ендпоінту балансу

#HTTP-ендпоінт для депозиту коштів на рахунок
@app.route("/deposit", methods=["POST"])
```

```
def deposit():
    #Ендпоінт приймає JSON дані з токеном та сумою депозиту
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    amount = data.get("amount")
    simulate_external_api_call()
    if not isinstance(amount, (int, float)) or amount <= 0:
        return jsonify({"error": "Invalid deposit amount"}), 400
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    account["balance"] += amount
    timestamp = datetime.datetime.now().isoformat()
    account["transactions"].append({
        "type": "deposit",
        "amount": amount,
        "timestamp": timestamp
    })
    log_transaction(account["username"], "deposit", amount)
    audit_log(f"Deposit of {amount} made by {account['username']}. New balance:
{account['balance']}")
    add_notification(account["username"], f"Deposit of {amount} successful. New
balance: {account['balance']}")
    #Інтегруємо транзакцію в блокчейн
    blockchain_log_transaction({
        "username": account["username"],
        "type": "deposit",
        "amount": amount,
        "timestamp": timestamp
    })
    return jsonify({"message": "Deposit successful", "new_balance":
account["balance"]}), 200
#Кінець ендпоінту депозиту

#Функція журналювання транзакцій
def log_transaction(username, transaction_type, amount):
    #Функція записує тип транзакції, суму та час проведення операції
    timestamp = datetime.datetime.now().isoformat()
    log_entry = {
        "username": username,
        "transaction_type": transaction_type,
        "amount": amount,
        "timestamp": timestamp
    }
    transaction_logs.append(log_entry)

#Кінець функції журналювання транзакцій

#HTTP-ендпоінт для зняття коштів з рахунку
@app.route("/withdraw", methods=["POST"])
def withdraw():
    #Ендпоінт приймає JSON дані з токеном та сумою зняття коштів
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    amount = data.get("amount")
    simulate_external_api_call()
    if not isinstance(amount, (int, float)) or amount <= 0:
        return jsonify({"error": "Invalid withdrawal amount"}), 400
    account = get_account_by_token(token)
```

```
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    if account["balance"] < amount:
        return jsonify({"error": "Insufficient funds"}), 400
    account["balance"] -= amount
    timestamp = datetime.datetime.now().isoformat()
    account["transactions"].append({
        "type": "withdraw",
        "amount": amount,
        "timestamp": timestamp
    })
    log_transaction(account["username"], "withdraw", amount)
    audit_log(f"Withdrawal of {amount} by {account['username']}. New balance:
{account['balance']}")
    add_notification(account["username"], f"Withdrawal of {amount} successful.
New balance: {account['balance']}")
    blockchain_log_transaction({
        "username": account["username"],
        "type": "withdraw",
        "amount": amount,
        "timestamp": timestamp
    })
    return jsonify({"message": "Withdrawal successful", "new_balance":
account["balance"]}), 200
#Кінець ендпоінту зняття коштів

#HTTP-ендпоінт для переказу коштів між рахунками
@app.route("/transfer", methods=["POST"])
def transfer():
    # Ендпоінт приймає JSON дані з токеном, цільовим
    # ім'ям користувача та сумою переказу
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    target_username = data.get("target_username")
    amount = data.get("amount")
    simulate_external_api_call()
    if not isinstance(amount, (int, float)) or amount <= 0:
        return jsonify({"error": "Invalid transfer amount"}), 400
    source_account = get_account_by_token(token)
    if source_account is None:
        return jsonify({"error": "Invalid token"}), 401
    if source_account["balance"] < amount:
        return jsonify({"error": "Insufficient funds"}), 400
    if target_username not in accounts:
        return jsonify({"error": "Target account does not exist"}), 400
    source_account["balance"] -= amount
    accounts[target_username]["balance"] += amount
    timestamp = datetime.datetime.now().isoformat()
    source_account["transactions"].append({
        "type": "transfer_out",
        "amount": amount,
        "target": target_username,
        "timestamp": timestamp
    })
    accounts[target_username]["transactions"].append({
        "type": "transfer_in",
        "amount": amount,
        "source": source_account["username"],
        "timestamp": timestamp
    })
    log_transaction(source_account["username"], "transfer_out", amount)
```

```
    log_transaction(target_username, "transfer_in", amount)
    audit_log(f"Transfer of {amount} from {source_account['username']} to
{target_username}.")
    add_notification(source_account["username"], f"Transfer of {amount} to
{target_username} completed.")
    add_notification(target_username, f"Received transfer of {amount} from
{source_account['username']}.")
    blockchain_log_transaction({
        "source": source_account["username"],
        "target": target_username,
        "type": "transfer",
        "amount": amount,
        "timestamp": timestamp
    })
    return jsonify({"message": "Transfer successful", "new_balance":
source_account["balance"]}), 200
#Кінець ендпоінту переказу коштів

#HTTP-ендпоінт для отримання історії транзакцій користувача
@app.route("/transactions", methods=["GET"])
def transactions():
    #Ендпоінт повертає список усіх транзакцій, здійснених користувачем
    token = request.headers.get("Authorization")
    simulate_external_api_call()
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
    return jsonify({"transactions": account["transactions"]}), 200
#Кінець ендпоінту історії транзакцій

#HTTP-ендпоінт для перевірки стану системи
@app.route("/health", methods=["GET"])
def health():
    #Ендпоінт повертає базову інформацію про роботу системи для моніторингу
    simulate_external_api_call()
    health_status = system_health_check()
    return jsonify(health_status), 200
#Кінець ендпоінту перевірки стану системи

#Функція для виконання перевірки стану системи
def system_health_check():
    #Функція повертає статус системи, час перевірки, кількість активних сесій та
кількість зареєстрованих облікових записів
    health_status = {
        "status": "OK",
        "time": datetime.datetime.now().isoformat(),
        "active_sessions": len(sessions),
        "registered_accounts": len(accounts)
    }
    return health_status
#Кінець функції перевірки стану системи

#---------------------- НОТИФІКАЦІЙНА СИСТЕМА ----------------------#

#HTTP-ендпоінт для отримання нотифікацій користувача
@app.route("/notifications", methods=["GET"])
def get_notifications():
    #Ендпоінт перевіряє заголовок Authorization для отримання токену сесії
    token = request.headers.get("Authorization")
    simulate_external_api_call()
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
```

```
    username = account["username"]
    user_notifications = notifications.get(username, [])
    return jsonify({"notifications": user_notifications}), 200
#Кінець ендпоінту отримання нотифікацій

#HTTP-ендпоінт для очищення нотифікацій користувача
@app.route("/notifications/clear", methods=["POST"])
def clear_notifications():
    #Ендпоінт приймає JSON дані з токеном сесії
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    token = data.get("token")
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid or missing token"}), 401
    username = account["username"]
    notifications[username] = []
    add_notification(username, "Notifications cleared.")
    audit_log(f"Notifications cleared for username: {username}")
    return jsonify({"message": "Notifications cleared."}), 200
#Кінець ендпоінту очищення нотифікацій

#---------------------- ЕНДПОІНТ ДЛЯ БЛОКЧЕЙН РЕЄСТРУ ----------------------#

#HTTP-ендпоінт для отримання даних блокчейну
@app.route("/blockchain", methods=["GET"])
def get_blockchain():
    #Ендпоінт повертає повний список блоків у блокчейні
    simulate_external_api_call()
    return jsonify({"blockchain": blockchain}), 200
#Кінець ендпоінту блокчейну

#---------------------- ЕНДПОІНТ ДЛЯ РОЗШИРЕНОГО ЛОГУВАННЯ ТА АУДИТУ ----------

#HTTP-ендпоінт для отримання аудит-логів
@app.route("/audit_logs", methods=["GET"])
def get_audit_logs():
    #Ендпоінт повертає список аудит-логів для моніторингу діяльності системи
    simulate_external_api_call()
    return jsonify({"audit_logs": audit_logs}), 200
#Кінець ендпоінту аудит-логів

#---------------------- ДОДАТКОВІ ФУНКЦІЇ ФОНОВОГО ОБРОБКИ ------------------#

#Додаткова функція для симуляції фонової обробки завдань у системі
def background_job_processing():
    #Функція імітує виконання періодичних фоновых завдань, наприклад, обробку
черг транзакцій
    while True:
        simulate_external_api_call()
        #Фонове логування поточного стану транзакцій
        print("Background Job: Processing queued tasks...")
        time.sleep(5)
#Кінець функції фонового процесу

#Додаткова функція для симуляції планових завдань технічного обслуговування
def scheduled_tasks():
    #Функція виконує планові завдання, такі як логування поточного стану системи
та інші профілактичні операції
    while True:
        simulate_external_api_call()
        current_status = system_health_check()
```

```
        print("Scheduled Task - System Status:", current_status)
        time.sleep(10)
#Кінець функції планових завдань



#----------------------- ОСНОВНИЙ БЛОК ВИКОНАННЯ ПРОГРАМИ --------------------#

if __name__ == "__main__":
    #Перевірка наявності файлу аудит-логів, створення, якщо не існує
    if not os.path.exists("audit.log"):
        open("audit.log", "w", encoding="utf-8").close()
    #Ініціалізація демонстраційних даних облікових записів
    initialize_dummy_data()
    #Створення генезис блоку для блокчейну
    create_genesis_block()
    #Вивід повідомлення про запуск системи
    print("Virtual ATM Cloud Service is starting up...")
    #Створення окремого потоку для фонового процесу обробки завдань
    bg_thread = threading.Thread(target=background_job_processing, daemon=True)
    bg_thread.start()
    #Створення окремого потоку для виконання планових завдань
    scheduled_thread = threading.Thread(target=scheduled_tasks, daemon=True)
    scheduled_thread.start()
    #Запуск Flask додатку на порту 5000
    app.run(host="0.0.0.0", port=5000, debug=False)
#Кінець реалізації розширеного функціоналу системи хмарного сервісу SaaS для
віртуальної АТМ
```

Файл data_text.py

```
#!/usr/bin/env python3
#Імпортуємо необхідні бібліотеки для роботи з веб-сервісом,
# шифруванням та обробкою даних
import uuid
import datetime
import time
import threading
import json
from flask import Flask, request, jsonify
from cryptography.fernet import Fernet

#Ініціалізація Flask додатку
app = Flask(__name__)

#Глобальні словники та списки для зберігання даних системи
accounts = {}           # Словник з інформацією про користувачів
sessions = {}           # Словник для зберігання сесій користувачів
transaction_logs = []   # Список журналу транзакцій
blockchain = []         # Список блоків блокчейн-реєстру
audit_logs = []         # Список аудит-логів

#Генеруємо ключ для шифрування та ініціалізуємо об’єкт Fernet для
криптографічних операцій
encryption_key = Fernet.generate_key()
cipher_suite = Fernet(encryption_key)

#---------------------- ФУНКЦІЇ ДЛЯ ШИФРУВАННЯ ДАНИХ ----------------------#

#Функція для шифрування тексту за допомогою Fernet
def encrypt_text(plain_text):
#Шифруємо вхідний текст та повертаємо рядок у вигляді зашифрованих даних
    encrypted_bytes = cipher_suite.encrypt(plain_text.encode())
    return encrypted_bytes.decode()
#Кінець функції шифрування тексту

#Функція для дешифрування тексту за допомогою Fernet
def decrypt_text(encrypted_text):
#Дешифруємо вхідний зашифрований текст та повертаємо вихідний рядок
    decrypted_bytes = cipher_suite.decrypt(encrypted_text.encode())
    return decrypted_bytes.decode()
#Кінець функції дешифрування тексту

#---------------------- ДОДАТКОВІ ФУНКЦІЇ ----------------------#

#Функція для отримання даних облікового запису за токеном сесії
def get_account_by_token(token):
#За допомогою токену повертаємо відповідний обліковий запис користувача, якщо
він існує
    if token in sessions:
        username = sessions[token]
        if username in accounts:
            return accounts[username]
    return None
#Кінець функції отримання облікового запису

#Функція для запису аудит-логів у глобальний список та файл
def audit_log(event):
#Логуємо події з зазначенням часу та записуємо інформацію у файл аудит-логів
    timestamp = datetime.datetime.now().isoformat()
    log_entry = {"event": event, "timestamp": timestamp}
    audit_logs.append(log_entry)
```

```
    with open("audit.log", "a", encoding="utf-8") as f:
        f.write(f"{timestamp} - {event}\n")
#Кінець функції аудит-логування

#---------------------- РОЗШИРЕНЕ УПРАВЛІННЯ ПРОФІЛЕМ ----------------------#

#HTTP-ендпоінт для перегляду профілю користувача
@app.route("/profile", methods=["GET"])
def view_profile():
#Отримуємо токен сесії з заголовку та повертаємо дані профілю користувача
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    profile_data = {
        "username": account.get("username"),
        "email": account.get("email", "Not Provided"),
        "fullname": account.get("fullname", "Not Provided"),
        "created_at": account.get("created_at", "N/A")
    }
    return jsonify({"profile": profile_data}), 200
#Кінець ендпоінту перегляду профілю

#HTTP-ендпоінт для оновлення профілю користувача
@app.route("/profile/update", methods=["POST"])
def update_profile():
#Оновлюємо email та fullname користувача за допомогою даних з запиту
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    new_email = data.get("email")
    new_fullname = data.get("fullname")
    if new_email:
        account["email"] = new_email
    if new_fullname:
        account["fullname"] = new_fullname
    audit_log(f"Profile updated for user {account['username']}.")
    return jsonify({"message": "Profile updated successfully."}), 200
#Кінець ендпоінту оновлення профілю

#HTTP-ендпоінт для зміни пароля користувача
@app.route("/profile/change_password", methods=["POST"])
def change_password():
#Здійснюємо зміну пароля, перевіряючи поточний пароль та зберігаючи новий у
зашифрованому вигляді
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    current_password = data.get("current_password")
```

```
    new_password = data.get("new_password")
    if not current_password or not new_password:
        return jsonify({"error": "Current and new passwords required"}), 400
    stored_password_encrypted = account.get("password")
    stored_password = decrypt_text(stored_password_encrypted)
    if current_password != stored_password:
        return jsonify({"error": "Current password incorrect"}), 401
    encrypted_new_password = encrypt_text(new_password)
    account["password"] = encrypted_new_password
    audit_log(f"Password changed for user {account['username']}.")
    return jsonify({"message": "Password changed successfully."}), 200
#Кінець ендпоінту зміни пароля

#---------------------- РОЗШИРЕНА АНАЛІТИКА ТРАНЗАКЦІЙ ----------------------#

#HTTP-ендпоінт для отримання аналітики транзакцій користувача
@app.route("/analytics/transactions", methods=["GET"])
def transaction_analytics():
#Обчислюємо сумарні значення по депозитах, зняттях та переказах для користувача
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    account = get_account_by_token(token)
    if account is None:
        return jsonify({"error": "Invalid token"}), 401
    transactions = account.get("transactions", [])
    total_deposit = sum(t["amount"] for t in transactions if t.get("type") ==
"deposit")
    total_withdraw = sum(t["amount"] for t in transactions if t.get("type") ==
"withdraw")
    total_transfer_in = sum(t["amount"] for t in transactions if t.get("type")
== "transfer_in")
    total_transfer_out = sum(t["amount"] for t in transactions if t.get("type")
== "transfer_out")
    analytics = {
        "total_deposit": total_deposit,
        "total_withdraw": total_withdraw,
        "total_transfer_in": total_transfer_in,
        "total_transfer_out": total_transfer_out,
        "transaction_count": len(transactions)
    }
    return jsonify({"analytics": analytics}), 200
#Кінець ендпоінту аналітики транзакцій

#---------------------- ПАНЕЛЬ АДМІНІСТРАТОРА ----------------------#

#HTTP-ендпоінт для отримання списку користувачів (адміністраторський доступ)
@app.route("/admin/users", methods=["GET"])
def admin_list_users():
#Перевіряємо, чи має користувач адміністративні права, та повертаємо дані про
всіх користувачів
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    admin_account = get_account_by_token(token)
    if admin_account is None or admin_account.get("role") != "admin":
        return jsonify({"error": "Unauthorized access"}), 401
    user_list = []
    for username, details in accounts.items():
        user_info = {
            "username": username,
            "email": details.get("email", "N/A"),
            "balance": details.get("balance", 0.0),
```

```
            "created_at": details.get("created_at", "N/A")
        }
        user_list.append(user_info)
    return jsonify({"users": user_list}), 200
#Кінець ендпоінту отримання списку користувачів

#HTTP-ендпоінт для отримання всіх транзакцій системи (адміністраторський доступ)
@app.route("/admin/transactions", methods=["GET"])
def admin_list_transactions():
#Перевіряємо права адміністратора та повертаємо детальну інформацію про всі
транзакції користувачів
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    admin_account = get_account_by_token(token)
    if admin_account is None or admin_account.get("role") != "admin":
        return jsonify({"error": "Unauthorized access"}), 401
    all_transactions = []
    for user, details in accounts.items():
        user_transactions = details.get("transactions", [])
        for trans in user_transactions:
            trans_copy = trans.copy()
            trans_copy["username"] = user
            all_transactions.append(trans_copy)
    return jsonify({"transactions": all_transactions}), 200
#Кінець ендпоінту отримання транзакцій

#HTTP-ендпоінт для видалення користувача (адміністраторський доступ)
@app.route("/admin/delete_user", methods=["POST"])
def admin_delete_user():
#Адміністратор може видалити обліковий запис користувача за наданим ім'ям
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    admin_account = get_account_by_token(token)
    if admin_account is None or admin_account.get("role") != "admin":
        return jsonify({"error": "Unauthorized access"}), 401
    data = request.get_json()
    if not data:
        return jsonify({"error": "No input data provided"}), 400
    username_to_delete = data.get("username")
    if not username_to_delete or username_to_delete not in accounts:
        return jsonify({"error": "User not found"}), 404
    if username_to_delete == "admin":
        return jsonify({"error": "Cannot delete admin account"}), 400
    del accounts[username_to_delete]
    audit_log(f"Admin {admin_account['username']} deleted user
{username_to_delete}.")
    return jsonify({"message": f"User {username_to_delete} deleted
successfully."}), 200
#Кінець ендпоінту видалення користувача

#---------------------- ФУНКЦІЇ ДЛЯ ДЕМОНСТРАЦІЇ ШИФРУВАННЯ -----------------#

#HTTP-ендпоінт для шифрування вхідного тексту
@app.route("/encrypt", methods=["POST"])
def encrypt_endpoint():
#Приймаємо текст з запиту та повертаємо зашифрований варіант цього тексту
    data = request.get_json()
    if not data or "text" not in data:
        return jsonify({"error": "No text provided"}), 400
    plain_text = data.get("text")
    encrypted = encrypt_text(plain_text)
```

```
    return jsonify({"encrypted_text": encrypted}), 200
#Кінець ендпоінту шифрування тексту

#HTTP-ендпоінт для дешифрування вхідного тексту
@app.route("/decrypt", methods=["POST"])
def decrypt_endpoint():
#Приймаємо зашифрований текст з запиту та повертаємо розшифрований варіант
    data = request.get_json()
    if not data or "encrypted_text" not in data:
        return jsonify({"error": "No encrypted text provided"}), 400
    encrypted_text = data.get("encrypted_text")
    try:
        decrypted = decrypt_text(encrypted_text)
    except Exception as e:
        return jsonify({"error": "Decryption failed"}), 400
    return jsonify({"decrypted_text": decrypted}), 200
#Кінець ендпоінту дешифрування тексту

#-- АВТОМАТИЧНЕ РЕЗЕРВНЕ КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ ДАНИХ --------------------#

#Функція для створення резервної копії даних системи
def backup_data():
#Формуємо об’єкт резервної копії з основних глобальних даних та записуємо його у
файл у форматі JSON
    backup = {
        "accounts": accounts,
        "sessions": sessions,
        "transaction_logs": transaction_logs,
        "blockchain": blockchain
    }
    with open("backup.json", "w", encoding="utf-8") as f:
        json.dump(backup, f, ensure_ascii=False, indent=4)
    audit_log("System backup created.")
    return True
#Кінець функції створення резервної копії

#Функція для відновлення даних системи з резервної копії
def restore_data():
#Зчитуємо дані з файлу резервної копії та відновлюємо глобальні змінні системи
    global accounts, sessions, transaction_logs, blockchain
    try:
        with open("backup.json", "r", encoding="utf-8") as f:
            backup = json.load(f)
        accounts = backup.get("accounts", {})
        sessions = backup.get("sessions", {})
        transaction_logs = backup.get("transaction_logs", [])
        blockchain = backup.get("blockchain", [])
        audit_log("System restored from backup.")
        return True
    except Exception as e:
        audit_log("Backup restoration failed.")
        return False
#Кінець функції відновлення даних

#HTTP-ендпоінт для створення резервної копії (адміністраторський доступ)
@app.route("/backup", methods=["GET"])
def backup_endpoint():
#Перевіряємо права адміністратора та створюємо резервну копію системи
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    admin_account = get_account_by_token(token)
    if admin_account is None or admin_account.get("role") != "admin":
```

```
        return jsonify({"error": "Unauthorized access"}), 401
    if backup_data():
        return jsonify({"message": "Backup created successfully."}), 200
    else:
        return jsonify({"error": "Backup failed."}), 500
#Кінець ендпоінту створення резервної копії

#HTTP-ендпоінт для відновлення даних з резервної копії (адміністраторський
доступ)
@app.route("/restore", methods=["POST"])
def restore_endpoint():
#Перевіряємо права адміністратора та відновлюємо дані системи з резервного файлу
    token = request.headers.get("Authorization")
    if not token:
        return jsonify({"error": "Missing token"}), 400
    admin_account = get_account_by_token(token)
    if admin_account is None or admin_account.get("role") != "admin":
        return jsonify({"error": "Unauthorized access"}), 401
    if restore_data():
        return jsonify({"message": "Data restored successfully."}), 200
    else:
        return jsonify({"error": "Restore failed."}), 500
#Кінець ендпоінту відновлення даних

#---------------------- ОСНОВНИЙ БЛОК ЗАПУСКУ СЕРВІСУ ----------------------#

if __name__ == "__main__":
#Запускаємо Flask додаток на порті 5001 з вимкненим режимом налагодження
    app.run(host="0.0.0.0", port=5001, debug=False)
#Кінець основного блоку запуску сервісу
```